AD-A104 079 NAVAL POSTGRADUATE SCHOOL MONTEREY CA F/G 9/2
ALTERATION OF THE CP/M-86 OPERATING SYSTEM.(U)
JUN 81 M B CANDALOR

UNCLASSIFIED NL

1 OF 2
AD
A104079

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

ALTERATION OF THE CP/M-86 OPERATING SYSTEM.

by

Michael Bruno Candalor

June 1981

Thesis Advisor: U. R. Kodres

Approved for public release; distribution unlimited

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A104 079
	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Alteration of the CP/M-86 Operating System	5. TYPE OF REPORT & PERIOD COVERED Master's Thesis: June 1981
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Michael Bruno Candalor	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940	12. REPORT DATE June 1981
	13. NUMBER OF PAGES 96
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

8086 Processor, operating system, microcomputer, BIOS interface, CP/M-86

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

CP/M-86 is a microcomputer (INTEL 8086) operating system developed and marketed by Digital Research. The operating system is designed so that a user can adapt the system to his own input/output hardware devices. This thesis develops interfaces to two floppy disk controllers, the iSBC 201 (single density) and the iSBC 202 (double density) controllers. The interface includes the writing of a boot loader embedded in the iSBC957 Execution Vehicle Monitor, the monitor system for the INTEL iSBC 86/12 single board computer. Also included is

an interface module for the cold start loader (loader BIOS). A design for the interface module of typical systems based on Winchester technology hard disks is also presented.

Accession For
[illegible] X
[illegible]
[illegible]
[illegible]

By

Distribution/

Availability Codes

Dist	Avail and/or Special
A	

Alteration of the CP/M-86 Operating System

by

Michael Bruno Candalor
Lieutenant Commander, United States Navy
B.S.M.E., United States Naval Academy, 1972

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1981

Author: Michael B Candalor
____

Approved by: Uno R. Kodres
____
Thesis Advisor

____
Second Reader

____
Chairman, Department of Computer Science

____
Dean of Information and Policy Sciences

ABSTRACT

CP/M-86 is a microcomputer (INTEL 8086) operating system developed and marketed by Digital Research. The operating system is designed so that a user can adapt the system to his own input/output hardware devices. This thesis develops interfaces to two floppy disk controllers, the iSBC 201 (single density) and the iSBC 202 (double density) controllers. The interface includes the writing of a boot loader embedded in the iSBC 957 Execution Vehicle Monitor, the monitor system for the INTEL iSBC 86/12 single board computer. Also included is an interface module for the cold start loader (loader BIOS) and an input and output interface, BIOS. A design for the interface module of typical systems based on Winchester technology hard disks is also presented.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

A. PURPOSE OF THIS THESIS

The adaptation of CP/M-86 to the hardware described herein was undertaken to provide an operating system for 8086 processor based single board computers at the Naval Postgraduate School. This operating system will support software development and system emulation for the AEGIS modeling project. The software will be available for general use at NPS. In addition the experience of modifying an operating system provided the author with an opportunity to learn about microcomputer hardware and microcomputer operating systems.

B. HISTORY OF MICROCOMPUTER OPERATING SYSTEMS

This is a brief overview of the history of microcomputer operating systems summarized from Ref. 1. It is necessarily brief as the advent of microcomputer operating systems is itself rather recent. Microcomputers came of age with the construction of the entire central processor on one chip, the replacement of core memory with inexpensive mass produced semiconductor memory, the availability of the floppy disk and the standardization of diskette format. At first, the primary applications of microcomputers were in real-time control systems such as machine controlled tools. In such applications, process management is the main thrust

and system I/O is negligible. This required a simple, customized operating system. The first microcomputer operating systems, more properly called executive systems, were for real time applications. As microcomputer systems became less expensive, it became possible to devote a system to a single user as a program development tool. This use presented the need for higher level language support, which meant that an operating system had to interface one or more programming language(s) to the hardware. Several microcomputer manufacturers have produced their own operating systems. These operating systems are specifically designed for a "computer system" and are generally not user configurable.

Unlike the large, powerful operating systems found in mainframe and large minicomputer timesharing systems, microcomputer operating systems are relatively austere and simple. One of the primary reasons for this difference is that a microcomputer is usually a single user system (with some exceptions). As a result, the operating system does not need to provide features such as memory protection, process scheduling and time sharing of the CPU(s). Besides the simpler interface required of a microcomputer operating system, the operating system and the applications programs must function in a small amount of primary storage, typically between 16K and 64K, as compared to several megabytes in the large mainframes. Even though relatively

small and simple, a microcomputer operating system must still provide file management, process management and I/O management.

Two representative microcomputer operating systems are INTEL's ISIS-II and Digital Research's CP/M-80. To operate under ISIS, the user requires a minimum of 32K of primary storage. The CP/M user requires a minimum of 16K. Both provide the basic functions required of an operating system. ISIS, however, will only run on an INTEL computer system configuration and is not user modifyable. CP/M-80 is designed to run on any 8080 or Z-80 based microcomputer system after the user has modified the program module containing the hardware dependencies. This factor alone makes CP/M popular and has resulted in the production of many CP/M compatible utility and application programs by other companies. ISIS has some features beyond those of CP/M in the area of development software for INTEL hardware. CP/M's dynamic debugger (DDT), however, is more powerful and easier to use than INTEL's ICE system. Both ISIS and CP/M support essentially the same file operations. Currently, because of its flexibility, CP/M is the most widely used microcomputer operating system.

Multi-user systems such as MP/M and microcomputer network systems such as CP/NET (both produced by Digital Research), are now available.

C. ADAPTATION TO THE USER'S ENVIRONMENT

Digital Research has attempted to make their CP/M operating systems as flexible, in terms of hardware suite, as possible. The method used is modular programming. The user interface, the Console Command Processor (CCP) has no hardware dependencies other than the CPU. The file management system, the Basic Disk Operating System (BDOS), is also independent of hardware. Both the CCP and the BDOS are interfaced to the Basic Input/Output System (BIOS) through logical I/O devices and logical disk devices. The BIOS, then, contains the logical device to physical device translation routines. Adaptation of the operating system to a unique environment requires only the modification of the appropriate BIOS routines, greatly simplifying the alteration process.

Once one has successfully completed one adaptation, follow-on adaptations will be much easier to achieve as an understanding of the operating system and its interface procedures is developed along with a better understanding of microcomputer architecture in general.

D. ORGANIZATION OF THIS THESIS

This thesis is organized as a blueprint for alteration of the CP/M-86 operating system to any specific hardware configuration. This methodology will also serve, at least in general, for the alteration of any operating

system-to-hardware interface. Chapter 1 is a brief introduction to microcomputer operating systems in general and the modification of the CP/M-86 operating system in particular. Chapter 2 reflects the investigation of the candidate operating system in order to understand how to adapt it to the existing hardware. Chapter 3 is a summary of the study of the typical floppy disk or Winchester technology disk and a look at possible hardware candidates. Chapter 4 covers the adaptation of the I/O interface module (BIOS) and the bootstrap program for these versions of the operating system. Chapter 5 discusses some of the difficulties encountered and a plan for adapting CP/M-86 to a hard disk. The appendices contain the programs developed as part of thesis and one of the programs which was used as a model.

II. STRUCTURE OF CP/M-86

A. OVERVIEW

CP/M-86 is a microcomputer operating system for INTEL CORPORATION'S 8086 processor based microcomputers. It is the logical successor to CP/M-80, a similar operating system developed and marketed by Digital Research for the INTEL 8080 processor. File compatibility has been preserved with all previous versions of CP/M. CP/M provides a general environment for program construction, storage, editing, execution and debugging. The file structure of version 2 of CP/M-80 is used, allowing as many as sixteen drives with up to eight megabytes on each drive.

CP/M-86 offers built-in utility commands, system transient commands and the capability of executing user defined transient commands (programs). Among the system transient programs are an Intel compatible assembler (ASM86) and a dynamic machine language program debugger (DDT). They are described in detail in Digital Research's publications [Ref. 2] and [Ref. 3] respectively.

A powerful feature of CP/M is its modularity. One of the three modules of the operating system, the Basic I/O System (BIOS), defines the hardware environment for the system. As a result of this modularity, CP/M-86 can be modified to run on any 8086/8088 processor based, single processor computer

system by merely changing the BIOS. A more detailed description of CP/M and its features is contained in Digital Research's publications [Ref. 4], [Ref. 5] and [Ref. 6].

B. ORGANIZATION OF CP/M-86

The sources of CP/M-86 information for this paper are [Ref. 4], [Ref. 5] and [Ref. 6]. This chapter freely summarizes the relevant material to this thesis.

The operating system is contained in file "CPM.SYS". "CPM.SYS" contains three program modules: the Console Command Processor (CCP), the Basic Disk Operating System (BDOS), and the user-configurable Basic Input/Output System (BIOS). This modularity allows the CCP and BDOS to be independent of the hardware in which the system is implemented.

The CCP is the system's interface to the user's console. It translates the user's commands into CP/M system calls in order to carry out the desired action. The BDOS module provides all the disk and file management. The BIOS contains all the hardware dependent features and interfaces. The operating system executes in any portion of memory above the interrupt locations, while the remainder of the address space is partitioned into as many as eight non-contiguous regions, as defined in a table in the BIOS.

CP/M-86 is too large a program to fit in the first two (system) tracks of a diskette. As a result the boot loader

loads into memory a cold start loader, called "LOADER.CMD", from the first two tracks. The boot loader makes the appropriate initializations and then transfers program control to the cold start loader. The cold start loader, which is essentially a subset of "CPM.SYS", finds "CPM.SYS" on the system disk, loads it into memory, makes the proper initializations, and finally transfers control to the operating system.

C. CCP BUILT-IN & TRANSIENT COMMANDS

The operation of CP/M-86 is similar to that of CP/M-80. Upon cold start the operating system signs on and drive A is logged-in, CP/M-86 then waits for an input command line. There are five built in commands:

DIR - displays the directory of the designated drive

ERA - erases the specified directory entry on the designated drive

REN - renames the designated file

TYPE - types the designated file to the logical console device

USER - changes user directories in multi-directory systems

Also the command line may begin with the name of a transient program with the assumed file type of CMD. CMD stands for "command file" and is used to differentiate CP/M-86 transient command files from COM files under CP/M-80

which serve the same purpose. Transient programs are loaded into memory in the Transient Program Area(s) (TPA), as defined in the BIOS, in stack order.

CP/M-86 supports programs written in three memory models: the 8080 model, the Small model and the Compact model.

The 8080 model supports programs which are directly translated from CP/M-80 when code and data areas are intermixed. The model consists only of a code group which, in turn, is normally a single segment of 64K or less. The operating system and the cold start loader are written in the 8080 model.

The Small model supports programs where there is a separate code and data group. Normally the Small model programs are 64K or less.

The Compact model occurs when any of the extra, stack or auxiliary groups are present in the program. Each group may consist of one or more segments.

The three models differ primarily in the manner in which the segment registers are initialized upon transient program loading. The operating system's program load function determines the memory model used by the transient program by examining the program group used. All three models are discussed in more detail in the next section.

1. Transient Program Execution Models

The initial values of the segment registers are determined by the "memory model" of the transient program and are described in the CMD file header generated by the program "GENCMD.CMD" or "GENCMD.COM". The three models are depicted in Figure 1.

8080 Model	Code and Data Groups Overlap
Small Model	Independent Code & Data Groups
Compact Model	Three or More Independent Groups

Figure 1 Transient Program Memory Models.

a. The 8080 Model

The 8080 Model is assumed when the transient program contains only a code group (containing both code and data). In such cases, the CS, DS and ES registers are all initialized to the beginning of the code group, while the SS and SP registers remain set to a 96-byte stack area in the CCP. The Instruction Pointer (IP) is set to 100H, similar to CP/M-80. The intermixed code and data regions are indistinguishable. This model allows simple translation of 8080, 8085 and Z80 code into the 8086 and 8088 environment. Following program load, the 8080 Model appears as in Figure 2, where low addresses are shown at the top of the diagram.

```
             SS: --------------
                 !            !
                 !    CCP     !
                 !            !
                 --------------
        SS + SP:

    CS, DS, ES: --------------
    DS + 0000H  !    base     !
                !    page     !
                --------------
    CS + 0100H: ! IP = 0100H  !
                !    code     !
                --------------
                !    data     !
                --------------
                    . . .
                --------------
                !    code     !
                --------------
                !    data     !
                --------------
```

Figure 2 The 8080 Memory Model.

b. The Small Model

The Small Model is assumed when the transient program uses both a code and data group. (In ASM86, all code is generated following a CSEG directive, while data is defined following a DSEG directive.) In this case CS is set to the beginning of the code group, the DS and ES registers are set to the start of the data group, and the SS and SP registers remain in the CCP's area as shown graphically in Figure 3.

```
           SS: --------------
               !     CCP    !
               !            !
               --------------
               !    Stack   !
               --------------
      SS + SP:

               --------------
               ! IP = 0000H !
               !    code    !
               !            !
       DS, ES: --------------
               !    base    !
               !    page    !
               --------------
  DS + 0100H:  !            !
               !    data    !
               !            !
               --------------
```

Figure 3 The Small Memory Model.

c. The Compact Model

The Compact Model is assumed when separate code and data groups are present, along with one or more of the remaining groups. In this case, the CS, DS and ES registers are initialized to the base address of their respective areas. The SS and SP registers remain in the CCP area. If the user intends to use the stack group as a stack area, the transient program must set the SS and SP registers upon entry. The initial configuration of the segment registers in this model is shown in Figure 4.

```
         SS:  -------------
              !           !
              !    CCP    !
              !           !
              -------------
              !   Stack   !
              -------------
    SS + SP:

         CS:  -------------
              !           !
              ! IP = 0000H !
              !   code    !
              !           !
         DS:  -------------
              !   base    !
              !   page    !
              -------------
DS + 0100H:   !           !
              !   data    !
              !           !
         ES:  -------------
              !           !
              !   data    !
              !           !
              -------------
```

Figure 4 The Compact Memory Model.

The values of the various segment registers can be programmatically changed during execution by changing the values in the base page as described in the preliminary documentation, thus allowing access to the entire memory space.

2. <u>Transient Program Setup And Termination</u>

Similar to CP/M-80, the CCP parses up to two file names following the command and places the properly formatted File Control Blocks (FCB's) at locations 005CH and 006CH in the base page relative to the DS register. Under CP/M-80, the default DMA (direct memory access)

address is initialized to 0080H in the base page. Due to the segmented memory of the 8086 and 8088 processors, the DMA address is divided into two parts: the DMA segment address and the DMA offset. Also, under CP/M-86, the default DMA base is set to the value of DS, and the default DMA offset is initialized to 0080H. Thus, CP/M-80 and CP/M-86 operate in the same way in that they both assume the default DMA address is the second half of the base page.

The CCP transfers control to the transient program through an 8086 "Far Call." In all but one case of the Compact Model, the transient program may choose to use the 96-byte CCP stack, and optionally return directly to the CCP upon program termination by executing a "Far Return." Programmatic termination also occurs when BDOS function zero is executed. The operator may terminate program execution by typing a single CONTROL-C during line edited input. This has the same effect as programmatic execution of BDOS function zero. Contrary to the operation of CP/M-80, no disk reset occurs and the CCP and BDOS modules are not reloaded from the disk upon program termination. In short, for the user familiar with CP/M-80, the CP/M-86 environment is very similar, but more powerful.

D. BDOS SUMMARY

Entry into the BDOS is made through the 8086 software interrupt # 224. The BDOS is, essentially, a set of 59

functions of three basic types; simple functions, file operations and extended operations. The interface convention for BDOS calls requires that function code be passed in register CL with parameters passed in register DL or DX depending on whether it is a byte or word value. Byte values are returned in the AL register and word values in registers AX and BX. Table 1 below, from Reference 6, summarizes these conventions. A full description of each BDOS function is given in [Ref.6].

BDOS Entry Registers	BDOS Return Registers
CX Function Code	AL Byte Value
DL Byte Parameter	AX Word Value
DX Word Parameter	BX Word Value
DS Data Segment	BX Double Word Offset
	ES Segment Address

Table 1 BDOS Parameter Conventions.

E. BIOS SUMMARY

The BIOS is loaded into memory just above the CCP and BDOS modules as illustrated in Figure 5.

Since the BIOS may be configured by the user, it may vary somewhat in length. Individual routines within the BIOS may be at different memory locations. In order to standardize the interface to the BIOS, all accesses to the BIOS are made through the jump vector at the beginning of that module. The BIOS, like the BDOS, also has parameter

```
                     ------------------------
CS, DS, ES, SS:!                        !
               !                        !
               ! Console                !
               !    Command             !
               !       Processor        !
               !                        !
               !          and           !
               ! Basic                  !
               !    Disk                !
               !       Operating        !
               !            System      !
 ---------     ------------------------
 ! CS + 2500H:! BIOS Jump Vector        !
 !             ------------------------
 ! CS + 253FH:!                         !
 !            !  BIOS Entry Points      !
 !            !                         !
 !             ------------------------
BIOS: !       ! Disk                    !
 !            !    Parameter            !
 !            !       Tables            !
 !             ------------------------
 !            ! Uninitialized           !
 !            !    Scratch RAM          !
 ---------     ------------------------
```

Figure 5 Memory Location of the BIOS.

passing conventions. Parameters for routines in the BIOS are passed in the CX register and the DX register when required. Byte values are returned in the BL register and word values in BX.

There are three major types of routines in the BIOS: system initialization/reinitialization, simple character I/O and disk I/O. All simple character I/O operations are assumed to be in ASCII, both upper and lower case, with the high order (parity) bit set to zero. CP/M sees all peripheral devices as "logical" devices. Translation from logical device selection to physical device assignment is

accomplished in the BIOS, thus isolating the CCP and BDOS from hardware dependencies. BIOS routine entry is explained in Digital Research's publication [Ref. 6]. The BIOS also contains the Disk Parameter Tables which contain the description of the disk drive and provide a scratchpad area for certain BDOS operations.

III. INPUT/OUTPUT DEVICES

In CP/M-86 the CCP and BDOS accomplish all I/O via four "logical" devices. The BIOS assigns whatever physical devices are in that particular system to those logical devices. This mapping in the BIOS preserves the independence of the CCP and BDOS from the hardware configuration.

A. LOGICAL I/O DEVICES

CP/M-86 addresses four logical I/O devices: the console, the list device, the punch device and the reader. The console is the principal interactive peripheral through which the operating system communicates with the operator. The list device is the principal listing device, usually a hardcopy printer. The punch device is the principal tape punching device, usually a high-speed paper tape punch or teletype. The reader is the principal tape reading device. When the "IOBYTE" function is implemented, dynamic logical to physical device mapping may be accomplished as described in Ref. 6.

B. PHYSICAL I/O DEVICES

The CONIN, CONOUT, LISTOUT, PUNCH and READER routines in the BIOS define the physical interfaces with peripherals. The system adapter may define, in the BIOS, such devices as cassette tape recorders etc. so long as it is interfaced

with one of the logical devices. In this adaptation the list device and the console device are both mapped to the serial edge connector where the CRT console is connected. The reader is "stubbed" with an "end of file" input, that is, instead of a routine to interface a physical read device, the BIOS simply returns an indication that the read has been complete. And the punch device map is "stubbed" with a return statement.

C. DISK DEVICES

1. Hard Disks, Floppy Disks

There are many implementations of the hard disk technologies. There are fixed and movable head disks, removable disk packs and even combination hard and floppy systems. Floppy diskettes come mainly in the 5" and 8" size, single and double density, single and double sided, and as indicated above in combination with hard disks.

2. Organization of Data

Although each disk drive may be different, data is stored in conceptually the same manner. The disk surface is divided into tracks (or cylinders, if a multi-platter system.) Each track is divided into sectors. Each sector is addressable by the controller, making it the basic unit of storage. In multi-platter and/or multi-head systems, to access the disk the controller must select the proper head/platter as well as the track and sector required.

The amount of data that can be stored on a device is dependent on the size of the device and the recording format. Double density, as the name implies, gives twice as much storage on a diskette as single density. The cost, however, is greater.

Although the basic unit of storage is the sector, sectors are not the same size in every system. In general, the larger the sector, the more efficient the storage, but the less efficient the access. Many systems allow the user to select the sector size from a limited set of choices. Sectors are normally a multiple of 128 bytes.

3. <u>Interfaces to the Computer</u>

The key to the storage of information on the recording media, at least from the operating system modifier's point of view, is the disk drive controller. The controller itself is usually a microprogrammed microprocessor. The controller handles the actual reading from and writing to the disk in addition to other functions such as seek, format etc. The relative autonomy of the controller frees the operating system from having to handle disk I/O on a primitive level. However, the BIOS, which is hardware specific, must still communicate with the controller at a fairly low level.

Most microcomputer system I/O is done by DMA. In general the host operating system creates, somewhere in memory, an entity, often called a "command packet" or "I/O

parameter block" or some similarly descriptive name. The "packet" is usually seven to ten bytes of information which contain the detailed command for the disk drive controller. These "packets" form the sole means of issuing I/O commands to the controller.

Normally the disk drive controller/interface shares a bus with the host system. As a result the controller's command/status registers have device addresses from the bus. In most systems, they can be set by the user prior to system start-up.

The host system sends the address of the I/O command packet to the command registers of the controller. Upon receipt of this address the controller initiates action to gain control of the bus. When the controller has control of the bus it reads the appropriate number of bytes from the address it was given. The controller decodes this information and then carries out the prescribed operation. The controller may signal completion in various ways, the most common being entering a completion code in the command packet for the host to read, sending an interrupt to the host processor, or storing the status in an on-board status register for the host to read.

Many systems allow the DMA to be "throttled", that is, the controller gives up control of the bus periodically in order to increase overall system speed.

Other features commonly included in disk drive controllers are: linked I/O, that is, the ability to execute more than one I/O command packet without prompting from the host processor. Multiple sector I/O, that is, the ability to read or write more than one sector in response to a single I/O command packet.

4. <u>Examples of Particular Controllers</u>

a. iSBC 201 (Single Density MDS)

The iSBC 201, as described in Ref. 7, is the controller/interface for INTEL's INTELLEC MDS 800, an 8080 processor based microcomputer development system.

(1) <u>iSBC 201 Controller Operation</u>. The controller is composed of two circuit boards, a channel board and an interface board. They interface with the host processor via the system MULTIBUS, a system's bus used by INTEL Corporation. The channel board and interface board together handle all communications between the host CPU and the diskette system. They contain an 8-bit microprogrammed processor which can access system memory for obtaining channel commands via DMA. The controller also monitors the disk subsystem status and error conditions and makes their status available to the host CPU.

This diskette system records data by the Frequency Modulation (FM) method, giving a formatted 8" diskette capacity of approximately 256K bytes, divided into 77 tracks of 26 sectors each.

Functionally, the host CPU must create a command packet in memory for each operation. INTEL calls this packet an I/O Parameter Block (IOPB). An IOPB is ten bytes in length and specifies all the details of the diskette operation to be performed. The CPU, in the case of CP/M-86, through the BIOS module, sends the address of the IOPB to the controller. Then the controller gains control of the bus, retrieves the IOPB and executes the command. Upon completion the controller posts the diskette subsystem status and, if enabled by the IOPB, sends a completion interrupt to the host CPU. The information in the IOPB consists of:

Byte 1 - the channel word, this byte specifies the enabling of the lock override, random format of the lock override, random format sequence, interrupt control, data word length, successor bit, branch on wait and wait bits.

Byte 2 - specifies the drive selected, data length (8 or 16 bits/word) and the operation to be performed.

Byte 3 - specifies the number of sectors to be transferred.

Byte 4 - specifies the target track number (Ø-77).

Byte 5 - specifies the first sector to be accessed (1-26).

Byte 6 - specifies the least significant byte of the buffer address.

Byte 7 - specifies the most significant byte of the buffer adress.

Byte 8 - indicates a block number which allows a unique identification of an IOPB during linked IOPB operations.

Byte 9 - contains the least significant byte of the buffer address of the next linked IOPB.

Byte 10 - contains the most significant byte of the buffer address of the next linked IOPB.

The iSBC 201 can execute seven commands:

1) recalibrate (seek track 0)
2) seek
3) format a track
4) write data (without address marks)
5) write data
6) read data
7) verify CRC

The controller has seven registers that are accessible to the host CPU. The host CPU can read three of the registers: The Result Status register indicates the status of both drives (ready or not ready), the status of the controller for that drive (present or not present), and the status of the controller's interrupt flip-flop

(interrupt pending or completed). The Result Type register indicates whether the Result Byte register contains I/O error codes or ready status. The Result Byte holds the I/O error codes or diskette drive status. The host CPU can write to four of the controller's registers: Writing anything to the Reset Diskette System register resets the entire diskette subsystem. Writing to the Stop Diskette Operation register terminates I/O after completion of the current operation. The Memory Address Lower register receives the least significant byte of the address of the IOPB. The Memory Address Upper register receives the most significant byte of the IOPB address and when written into signals the controller to retrieve the IOPB and commence the specified operation.

(2) BIOS Use of the iSBC 201. The CP/M-86 BIOS uses only operations 1, 5 and 6 (seek is implicit in read and write operations). In addition, CP/M-86 does not use linked IOPB's and only does single sector disk accesses. This very much simplifies the I/O routines in the BDOS and the BIOS. Not using the linked IOPB capability allows reducing the IOPBs to the first seven bytes, of which bytes 1 and 3 remain constant. Byte One remains unchanged because the mode of disk access remains unchanged. Byte Three, the number of sectors, remains set at one, and the operating system is freed from computing the number of sectors per

access. These simplifications allow the BIOS to have a single IOPB template in memory.

A limitation of the iSBC 201 is its 16-bit addressing. This limitation means that the controller can only address 64K of system memory as compared to the 8086 processor's megabyte of address space. As a result, the external address of the iSBC 86/12 must reside in the first 64K of the megabyte (from 00000H to 0FFFFH). The BIOS in this adaptation converts the segment and offset address provided by the BDOS into a 16-bit physical address for the controller.

(3) Bootstrap Use of the iSBC 201. The bootstrap program does use the multi-sector access capability of the controller for loading the cold start loader. This requires four IOPBs in the bootstrap program but reduces the number of disk accesses from 53 to four. Considering the specialized function of the bootstrap loader and its lack of interface with the BDOS, this is a very efficient deviation from the otherwise efficient CP/M method of disk access.

b. iSBC 202 (Double Density MDS)

The iSBC 202 is the controller/interface for INTEL's INTELLEC MDS 888 microcomputer development system. It is described fully in Ref. 8.

(1) iSBC 202 Controller Operation. From the users point of view this controller is essentially the same as the iSBC 201. The main difference is the recording

format. Modified-Modified Frequency Modulation (MMFM) is used, allowing the same media to hold (formatted) 512K bytes of data, divided into 77 tracks of 52 sectors each. This is twice the capacity of the single density system.

(2) <u>BIOS Use of the iSBC 202</u>. The interface to the controller is the same as that of the iSBC 201. The difference in organization and capacity is only evident in the disk definition table "DOUBLE.LIB".

(3) <u>Bootstrap Use of the iSBC 202</u>. CP/M's double density formatter formats the first two tracks of a diskette in single density, ie. 26 sectors per track. The cold start loader fits in the first two tracks of a double density in the same way as in single density. As a result, the same bootstrap program will load the cold start loader from both single and double density diskettes.

c. REMEX RDW 3200

The RDW 3200, as described by Ref. 9, Ref. 10 and Ref. 11, is a multi drive unit consisting of a fixed Winchester Technology 14" disk and two 8" flexible diskette drives. The diskette drives are "jumper" selected as either single or double density. In both types the sector size is selectable. The formatted capacity of the fixed disk with sector size set at 128 bytes is 10 megabytes. This data is on 210 tracks of 124 sectors for each of two read/write heads. The single density floppy drives, formatted for 128 bytes per sector, hold 26 sectors on each of 77 tracks for a

total of 256K bytes of storage. Set for double density, the smallest sector size available is 256 bytes. At 26 sectors per track, for 77 tracks, formatted storage is 512K bytes. If this drive were used for CP/M-86 in the double density mode, the difference between diskette sector size (256 bytes) and CP/M-86 sector size (128 bytes) would be handled by a "blocking/deblocking" algorithm like the one provided with CP/M-86.

(1) The RDW Controller. The heart of the controller is a microprogrammed Motorola 6820 8-bit microprocessor. The controller physically resides inside the RDW frame and is linked to the host system by an interface card. This alteration utilized a MULTIBUS interface, which resided in the host's system MULTIBUS. The interface provides registers for communication between the host and the controller CPU's. Data can be handled as 8-bit words, 16-bit words or as 8-bit half-words. The controller can accomplish I/O by DMA, programmed I/O or by interrupts. All disk writes are by Modified-Modified Frequency Modulation (MMFM). The disk drive system can also be DMA throttled, which permits other masters to gain access to the system's bus in between accesses by the disk unit.

Functionally, the host CPU must create a command packet in memory for each operation. A command packet is six to fourteen bytes in length and specifies all the details of the disk operation to be performed. In the

DMA mode the host CPU must test the status register in the controller interface to assure that the controller is ready. When the controller is ready the CPU, in the case of CP/M-86, through the BIOS module, sends the address of the command packet to the controller interface. Then the controller gains control of the bus, retrieves the command packet and executes the command. Upon completion, the controller posts the disk subsystem status in the command packet in system memory and, if enabled by the command packet, sends a completion interrupt to the host CPU. The command packet consists of six to fourteen bytes. This controller supports five types of operations. The size of the packet and the information it contains are determined by the operation to be performed. The five operations supported are:

1) read data/write data
2) write I.D. and data for single record (fixed disk only)
3) copy from one drive to another
4) format designated disk
5) maintenance package

The controller has four registers that are accessible to the host CPU. The base address of these registers is switch selectable. The base address plus one is the status register, from which the host CPU determines

system status. The base address plus three receives the lower byte of the address of the command packet. The base address plus two receives the middle byte of the command packet address. The base address receives the upper byte of the packet address (RDW 3200 supports 24-bit addressing) and when written into signals the controller to start DMA.

(2) BIOS Use of the RDW 3200. The CP/M-86 BIOS would use only the read/write operation. The fact that the hard disk has more than one head would require that the BIOS disk definition table look like one continuous set of tracks and that prior to initiating DMA, the BIOS translate a logical track number to a physical head and track number. The read and write packets have the same format which requires only one packet template in the BIOS. That packet takes the following form; indicated as 16-bit words:

Word 0 - I/O modifiers (linked I/O,interrupts,etc.), operation and drive selected.

Word 1 - status word - written by controller.

Word 2 - track number.

Word 3 - head and sector start number.

Word 4 - lower 16 bits of DMA address.

Word 5 - high byte of DMA address.

Word 6 - transfer word count.

Although the RDW supports 24-bit addressing, it requires a 24-bit physical address, not the

segment and offset type address provided by the BIOS. Therefore the BIOS must translate the addresses before placing them in the command packet and before sending them to the interface.

(3) <u>Bootstrap Use of the RDW 3222</u>. The bootstrap program would use the multi-sector access capability of the controller for loading the cold start loader (the command packet specifies the number of words to be transferred). If the operating system were to be loaded from a diskette, the bootstrap operation would be very much like that described for the iSBC 201. For a system load from the hard disk the bootstrap program could load the operating system without the use of a cold start loader. This would only require two disk accesses, one to determine the load location and the other to actually load "CPM.SYS".

IV. ALTERATION OF CP/M-86

A. CHANGES REQUIRED TO IMPLEMENT CP/M-86

As distributed, CP/M-86 is set up for operation with an Intel SBC 86/12 microcomputer and an Intel SBC 204 diskette controller with a Shugart SA-800 floppy disk drive. Since CP/M-86 is modular, only the BIOS need be modified for "non standard" hardware. The distribution version includes source code for its BIOS and a skeletal BIOS to aid in the construction of a customized version. Although the distribution version does not provide a bootstrap ROM, the source code for the program is provided. This source code provided an example for the creation of a customized bootstrap program. The bootstrap ROM is available from Digital Research.

The changes required to customize the BIOS can be divided into four types. The first consideration is the computer selected for the implementation. If an 8086/8088 based computer other than the iSBC 86/12 were chosen, the computer initialization, including the constant definitions for USART ports and character I/O routines such as console status, console input and console output, have to be changed to match the host hardware. Since the iSBC 86/12 was used, no changes were required in this portion of the BIOS. Second, if the disk drive controller or other DMA device is

not an iSBC 204, the controller port definitions and the routines which actually communicate with the controller must be altered. The "execute" and "sendcom" routines were the bulk of the modification. These routines check system status, translate system commands to the language of the controller, deliver the commands to the hardware and handle any hardware errors. Third, if any other serial or parallel I/O device is to be used, the appropriate initialization and execution routines must be written. The fourth consideration is the disk definition table which is assembled with the BIOS via an "include" statement. Disk parameter tables must be created to describe the disk system. Disk parameter tables are discussed in the next section. In this version only the second and fourth types of modifications were necessary and those changes are reflected in Appendix A (single density) and Appendix B (double density). Appendix D contains the distribution BIOS. After assembling the BIOS, the hexadecimal code, "BIOS.H86", is appended to "CPM.H86" and a command file is generated by the method described in Ref. 6 using the GENCMD utility. The file created is named "CPM.SYS" and is the operating system.

B. DISK PARAMETER TABLES

The disk parameter table serves to define the organization of the storage media for the BDOS file management functions. The disk definition consists of the

sequence of statements in Figure 6 (as shown in Ref. 6). The DISKS statement defines the number of drives in the system, with n being an integer from 1 to 16. A series of DISKDEF statements follow. Each statement defines the characteristics of a logical disk, Ø through n-1. DISKDEF statements are formed as defined in Ref. 6. The format is shown in Figure 7.

```
DISKS    n
DISKDEF  Ø,...
DISKDEF  1,...
......
DISKDEF  n-1
......
ENDEF
```

Figure 6 BIOS Disk Definition File.

```
DISKDEF dn,fsc,lsc,[skf],bls,dks,dir,cks,ofs,[Ø]
```

where

```
dn       is the logical disk number, Ø to n-1
fsc      is the first physical sector number (Ø or 1)
lsc      is the last sector number
skf      is the optional skew factor
bls      is the data allocation block size
dks      is the disk size in bls units
dir      is the number of directory entries
cks      is the number of "checked" directory entries
ofs      is the track offset to logical track ØØ
[Ø]      is an optional 1.4 compatibility flag
```

Figure 7 DISKDEF Statement Format.

The disk tables may be generated by hand or by executing the GENDEF utility program. The table provided with the

distribution version, called "SINGLES.LIB", was generated from the source file "SINGLES.DEF" by the GENDEF utility running under CP/M-80. This table was correct for the single density implementation. It was necessary to create a new table for the double density system. This file is called DOUBLE.DEF. Table generation is described fully in Section 6 of Ref. 5. The disk parameter tables are listed in the BIOS right after the "include" statement (see Appendices A and B).

C. COLD START

1. The Cold Start Loader

Since CP/M-86 is too large to fit in the first two (system) tracks of a diskette, it is loaded into memory in two st ps. First, a cold start loader is loaded from the first two tracks into memory. Next the loader loads the operating system and transfers control to it. The loader ("LOADER.CMD") is a simplified version of CP/M-86 with enough power to locate the operating system file "CPM.SYS" on the current disk, make the proper initializations, load CP/M-86 into memory and then transfer program control to it. The loader is created from files LDCPM, IDBDOS and the loader version of the BIOS. The loader BIOS is generated from the same source code as the BIOS by setting the software switch "LOADER_BIOS" equal to true prior to assembly.

The loader program is moved to the first two tracks of a diskette by the LDCOPY utility if running on a working CP/M-86 system. If developement is done on a CP/M-80 system this can be accomplished with the DDT and SYSGEN utilities. Ref. 6 errs in its discription of the latter procedure. The correct procedure is described in the next chapter.

2. <u>The Bootstrap ROM</u>

In order to get the cold start loader into memory, there must be a bootstrap loader of some kind. This boot loader must initialize the programmable chips on the single board computer and the disk drive controller which will access the operating system disk. It then loads the first two tracks of the diskette in the system disk drive into memory and then transfers control to the program loaded, "LOADER.CMD". The bootstrap program is normally resident in a read only memory (ROM) or electrically programmable ROM (EPROM) and is then referenced to as the boot ROM.

The distribution version of CP/M-86 also contains the listing for a bootstrap ROM (ROM.A86). The boot ROM itself is available from Digital Research. When installed, it becomes part of the 8086 address space. Upon system reset, the processor begins execution at effective address 0FF000H, which is the top paragraph of the iSBC 86/12 EPROM space. The bootstrap program is hardware dependent which necessitated the creation of a customized initial loader for this implementation.

Intel's SBC 957 Execution Vehicle Monitor (EVM) occupies the EPROM locations when installed in the iSBC 86/12 and is currently in use at the Naval Postgraduate School. In order to retain the use of the iSBC 957 and to simplify implementation, the customized bootstrap program has been embedded in a free area of the EVM's EPROMs. Since the monitor initializes the single board computer when it is started, the CP/M-86 bootstrap task is simplified. The bootstrap program listing is in Appendix C. It is a modified version of the "debug" version of Digital Research's ROM program. The modified bootstrap program is located at effective address ØFFD40H. It may be executed from the EVM by executing the command GFFD4:Ø or its equivalent.

V. CONCLUSIONS AND RECOMMENDATIONS

A. ADAPTATION DIFFICULTY

Modification of CP/M-86 is a straightforward simple procedure if one is familiar with CP/M on a system's software level and with at least some representative hardware. If one does not have such a background (the author did not), the task is not overwhelming, but considerably more difficult. The novice will probably invest much time and effort in investigating "dead ends" because of not understanding the logical design of the operating system. A particularly vexing problem encountered in the first adaptation was that in the later stages of development, every error in the corrected software seemed to destroy the information on the diskette, making debugging difficult and requiring frequent regeneration of software. During this period of "destructive testing" approximately 90% of the time and effort were spent on such overhead and only 10% on actual debugging. The real problem there was not the time lost but the interruption in the train of thought.

Documentation inadequacies are another source of problems. The alteration guide for CP/M-86 provided by Digital Research (Ref. 6) assumed a thorough knowledge of CP/M-80, which was not possessed by the author. The CP/M-80 documentation also seemed to assume a thorough knowledge of

the operating system's modules. In addition, there were several errors in the alteration guide.

The procedure for moving the cold start loader to tracks zero and one under CP/M-80 is incorrect and if followed the first 800H bytes of the program will be lost. A correct procedure is to load the cold start loader with DDT, move the program so that it starts at 900H, exit DDT and finally call the SYSGEN utility. A correct sequence of commands looks like this:

```
DDT LOADER.CMD
m1100,1800,1900
ma00,1100,1200
m400,a00,c00
m100,400,900
<CONTROL-C>
SYSGEN
<CR>
B
<CR>
```

The documentation for the 8086 assemblers, "ASM86.COM" and "ASM86.CMD" also contains errors. According to the user's manual, [Ref. 2], the "device switch" for the listing device is "P". The correct switch is "Y".

The technical manuals provided with the disk drives and controllers used rather ambiguous and non standardized

terms. This often required experimentation to determine what was really meant.

Resolution of the above difficulties, however, was a good learning experience for the author.

B. RECOMMENDATIONS FOR FUTURE HARD DISK ADDITION

1. Discussion

Although there are several methods of accomplishing disk I/O, DMA seems to be the simplest to implement and debug. A future hard disk addition would greatly enhance CP/M-86's usefulness. In this vein, a hard disk/floppy disk combination would be ideal. The combination of hard and floppy disks would provide the speed and storage capacity on one hand (from the hard disk) and the ability for the user to keep copies of his files where he is assured of their security and integrity. However, inclusion of the iSBC 201 or 202 is not recommended. The limited addressing capability of these controllers would hinder overall system effectiveness and force the processor to operate in the bottom 64K of the address space. As a rule of thumb, if more than one device is to be added to the basic system, only one device should be added at a time.

2. Template for Adaptation

Given that a hard disk is to be installed in place of the diskette system, the following procedure should be followed:

First, the CP/M-86 BIOS should be studied in conjunction with the current hardware to see how the interface is currently accomplished. The system modifier must understand how the operating system interacts with hardware before creating his own interface. Second, the target hardware must be studied. The electronics are not important, but what the hardware does logically and how it communicates with the controller is paramount. In particular, the organization of data on a disk drive must be thoroughly understood. If the organization of data is selectable, the most efficient and straightforward organization must be chosen. If it is not selectable and not directly compatible with the BDOS, a "blocking/deblocking" or some other scheme must be considered. Third, a disk definition must be written to reflect the logical organization of the disk. If the logical organization of data does not match its physical organization, the executing routine in the BIOS would have to make the translation. For example, in a multi head disk system, the tracks would have to be numbered in the disk definition as though they were on the same platter (logical org.), the BDOS would select a sector and a "logical" track for I/O, but before sending the channel command the BIOS would have translate that "logical" track number to a head and track combination. Fourth, a template for the channel command should be placed in the BIOS with appropriate variable names to allow the BDOS to

provide as much information directly as possible. Fifth, write the "execute" routine. This routine, the bulk of the coding, must complete the channel command, prepare the disk for access, send the activating command, check completion status and handle hardware errors. This step requires a good knowledge of the target disk system and is very much dependent on the disk chosen. Sixth, once the revised BIOS is written, it must be assembled (in the loader version too, if booting from a floppy disk). The files "CPM.H86", "BIOS.H86" and "PAT2.H86" are combined into "CPMX.H86". This resulting file is converted to executable form by executing the command "GENCMD CPMX 8080[A40]" as described in Ref. 6. The resulting file is then renamed "CPM.SYS".

The bootstrap program will be very simple. It can be written to explicitly read the first sector of the disk, to determine the loading target address, and to read the following 76 (128 byte) sectors. Once the BIOS has been modified, the bootstrap program will be almost a trivial subset of that code.

APPENDIX A

```
        title   'Customized Basic I/O System'

;*********************************************
;*                                           *
;* This Customized BIOS adapts CP/M-86 to    *
;* the following hardware configuration      *
;*      Processor:  iSBC 8612                *
;*      Controller:  iSBC 201                *
;*      Memory model:  8080                  *
;*                                           *
;*      Programmer:  M.B. Candalor           *
;*      Revisions :                          *
;*                                           *
;*********************************************

true            equ -1
false           equ not true
cr              equ 0dh ;carriage return
lf              equ 0ah ;line feed
max_retries     equ 10  ;for disk i/o, before perm error

;*********************************************
;*                                           *
;* Loader_bios is true if assembling the     *
;* LOADER BIOS, otherwise BIOS is for the    *
;* CPM.SYS file.                             *
;*                                           *
;*********************************************

LOADER_BIOS     EQU TRUE
bdos_int        equ 224 ;reserved BDOS interrupt

        IF      not loader_bios
;---------------------------------------------
;|                                           |
bios_code       equ 2500h
ccp_offset      equ 0000h
bdos_ofst       equ 0B06h ;BDOS entry point
;|                                           |
;---------------------------------------------
        ENDIF   ;not loader_bios

        IF      loader_bios
;---------------------------------------------
;|                                           |
bios_code       equ 1200h ;start of LDBIOS
ccp_offset      equ 0003h ;base of CPMLOADER
bdos_ofst       equ 0406h ;stripped BDOS entry
```

```
;|                                              |
;-----------------------------------------------
        ENDIF   ;loader_bios

csts    equ 0dah        ;I8251 status port
cdata   equ 0d8h        ;   "      data
;
;
;***********************************************
;*                                             *
;* INTEL iSBC 201 Disk Controller Ports        *
;*                                             *
;***********************************************


base    equ     078h
rtype   equ     base+1
rbyte   equ     base+3
reset   equ     base+7

dstat   equ     base
ilow    equ     base+1
ihigh   equ     base+2

        cseg
        org     ccpoffset
ccp:
        org     bios_code

;***********************************************
;*                                             *
;* BIOS Jump Vector for Individual Routines    *
;*                                             *
;***********************************************

 jmp INIT       ;Enter from BOOT ROM or LOADER
 jmp WBOOT      ;Arrive here from BDOS call 0
 jmp CONST      ;return console keyboard status
 jmp CONIN      ;return console keyboard char
 jmp CONOUT     ;write char to console device
 jmp LISTOUT    ;write character to list device
 jmp PUNCH      ;write character to punch device
 jmp READER     ;return char from reader device
 jmp HOME       ;move to trk 00 on cur sel drive
 jmp SELDSK     ;select disk for next rd/write
 jmp SETTRK     ;set track for next rd/write
 jmp SETSEC     ;set sector for next rd/write
 jmp SETDMA     ;set offset for user buff (DMA)
 jmp READ       ;read a 128 byte sector
 jmp WRITE      ;write a 128 byte sector
 jmp LISTST     ;return list status
```

```
 jmp SECTRAN       ;xlate logical->physical sector
 jmp SETDMAB       ;set seg base for buff (DMA)
 jmp GETSEGT       ;return offset of Mem Desc Table
 jmp GETIOBF       ;return I/O map byte (IOBYTE)
 jmp SETIOBF       ;set I/O map byte (IOBYTE)

;*********************************************
;*                                           *
;* INIT Entry Point, Differs for LDBIOS and  *
;* BIOS, according to "Loader_Bios" value    *
;*                                           *
;*********************************************

INIT:   ;print signon message and initialize hardware
        mov ax,cs          ;we entered with a JMPF so use
        mov ss,ax          ;CS: as the initial value of SS:,
        mov ds,ax          ;DS:,
        mov es,ax          ;and ES:
        ;use local stack during initialization
        mov sp,offset stkbase
        cld                ;set forward direction

        IF      not loader_bios
;---------------------------------------------
;|                                           |
        ; This is a BIOS for the CPM.SYS file.
        ; Setup all interrupt vectors in low
        ; memory to address trap

        push ds            ;save the DS register
        mov IOBYTE,0       ;clear IOBYTE
        mov ax,0
        mov ds,ax
        mov es,ax          ;set ES and DS to zero
        ;setup interrupt 0 to address trap routine
        mov int0_offset,offset int_trap
        mov int0_segment,CS
        mov di,4
        mov si,0           ;then propagate
        mov cx,510         ;trap vector to
        rep movs ax,ax     ;all 256 interrupts
        ;BDOS offset to proper interrupt
        mov bdos_offset,bdos_ofst
        pop ds             ;restore the DS register

;       (additional CP/M-86 initialization)
;|                                           |
;---------------------------------------------
        ENDIF   ;not loader_bios

        IF      loader_bios
```

```
;-----------------------------------------
;|                                       |
        ;This is a BIOS for the LOADER
        push ds             ;save data segment
        mov ax,0
        mov ds,ax           ;point to segment zero
        ;BDOS interrupt offset
        mov bdos_offset,bdos_ofst
        mov bdos_segment,CS ;bdos interrupt segment
;       (additional LOADER initialization)
        pop ds              ;restore data segment
;|                                       |
;-----------------------------------------
        ENDIF   ;loader_bios

        mov bx,offset signon
        call pmsg           ;print signon message
        mov cl,0            ;default to dr A: on coldstart
        jmp ccp             ;jump to cold start entry of CCP

WBOOT:  jmp ccp+6           ;direct entry to CCP at command level

        IF      not loader_bios
;-----------------------------------------
;|                                       |
int_trap:
        cli                 ;block interrupts
        mov ax,cs
        mov ds,ax           ;get our data segment
        mov bx,offset int_trp
        call pmsg
        hlt                 ;hardstop
;|                                       |
;-----------------------------------------
        ENDIF   ;not loader_bios

;*****************************************
;*                                       *
;*   CP/M Character I/O Interface Routines   *
;*                                       *
;*    console is USART (i8251A) on iSBC 8612 *
;*               at ports D8/DA          *
;*****************************************

CONST:          ;console status
        in al,csts
        and al,2
        jz const_ret
        or al,255           ;return non-zero if rda
const_ret:
        ret                 ;rcvr data available
```

```
CONIN:                          ;console input
        call CONST
        jz CONIN                ;wait for RDA
        in al,cdata
        and al,7fh              ;read data & remove parity bit
        ret

CONOUT:         ;console output
        in al,csts
        and al,1                ;get console status
        jz CONOUT
        mov al,cl
        out cdata,al            ;transmitter buffer is empty
        ret                     ;then return data

LISTOUT:                        ;list device output
                                ;not implemented
        ret

LISTST:                         ;poll list status
                                ;not implemented
        ret

PUNCH:          ;write punch device
                                ;not implemented

READER:
        mov al,1ah              ;return eof for now
        ret

GETIOBF:
        MOV AL,0                ;IOBYTE NOT IMPLEMENTED
        ret

SETIOBF:
        ret                     ;iobyte not implemented

; Routine to get and echo a console character
;       and shift it to upper case

uconecho:
        call CONIN              ;get a console character
        push ax
        mov cl,al               ;save and
        call CONOUT
        pop ax                  ;echo to console
        cmp al,'a'
        jb uret                 ;less than 'a' is ok
        cmp al,'z'
        ja uret                 ;greater than 'z' is ok
```

```
        sub al,'a'-'A'   ;else shift to caps
uret:
        ret
pmsg:
        mov al,[BX]      ;get next char from message
        test al,al
        jz return        ;if zero return
        mov CL,AL
        call CONOUT      ;print it
        inc BX
        jmps pmsg        ;next character and loop

;*********************************************
;*                                           *
;*        Disk Input/Output Routines         *
;*                                           *
;*********************************************

SELDSK:           ;select disk given by register CL
ndisks  equu     2 ;number of disks (up to 16)
        mov disk,cl      ;save disk number
        mov bx,0000h     ;ready for error return
        cmp cl,ndisks    ;n beyond max disks?
        jnb return       ;return if so
        mov ch,0         ;double(n)
        mov bx,cx        ;bx = n
        mov cl,4         ;ready for *16
        shl bx,cl        ;n = n * 16
        mov cx,offset dpbase
        add bx,cx        ;dpbase + n * 16
return: ret              ;bx = .dph

HOME:   ;move selected disk to home position (Track 0)
        mov io_com,homcom
        mov trk,0
        call execute
        ret

SETTRK: ;set track address given by CL
        mov trk,CL
        ret

SETSEC: ;set sector number given by cl
        mov sect,CL
        ret

SECTRAN: ;translate sector CX using table at [DX]
        mov ch,0
        mov bx,cx
        add bx,dx        ;add sector to tran table address
        mov bl,[bx]      ;get logical sector
```

```
        ret

SETDMA: ;set DMA offset given by CX
        mov dma_adr,CX
        ret

SETDMAB: ;set DMA segment given by CX
        mov dma_seg,CX
        ret
;
GETSEGT:  ;return address of physical memory table
        mov bx,offset seg_table
        ret

;*********************************************
;*                                           *
;*  All disk I/O parameters are setup:       *
;*     DISK     is disk number      (SELDSK) *
;*     TRK      is track number     (SETTRK) *
;*     SECT     is sector number    (SETSEC) *
;*    DMA_ADR  is the DMA isb offset        *
;*  READ reads the selected sector to the DMA*
;*  address, and WRITE writes the data from  *
;*  the DMA address to the selected sector   *
;*                                           *
;*********************************************

READ:
        mov cl,4
        mov al,disk       ;combine disk selection
        sal al,cl         ;with opcode
        or al,rdcode
        mov io_com,al     ;create iopb
        jmps execute

WRITE:
        mov cl,4
        mov al,disk
        sal al,cl
        or al,wrcode      ;create iopb for write
        mov io_com,al

EXECUTE:

outer_retry:
        mov rtry_cnt,max_retries

retry:
        in al,rtype       ;clear controller
        in al,rbyte       ;
        call sendcom
```

```
idle:   in al,istat        ;wait for completion
        and al,4           ;ready
        jz idle

;       check i.o. completion ok
        in al,rtype
;       00 unlinked i/o complete        01 linked i/o comp
;       12 disk status changed          11 (not used)
;       must be a 00 in al
        test al,10h        ;ready status change?
        JNZ WREADY
        OR AL,0
        jnz werror         ;some other error, retry

;       check i/o error bits
        in al,rbyte
        rcl al,1
        mov err_code,80h
        jb wready          ;unit not ready
        rcr al,1
        mov err_code,al
        and al,0feh        ;any other errors?
        jnz werror
;
;       read or write is ok, al contains 0
        ret

wready: ;not ready, treat as an error for now
        in al,rbyte        ;clear result byte
        jmps trycount

werror: ;return hardware malfunction
trycount:
        dec rtry_cnt
        jnz retry
        mov al,err_code
        mov ah,0
        mov bx,ax          ;make error code 16 bits
        mov bx,errtbl[BX]
        call pmsg          ;print appropriate message
        in al,cdata        ;flush usart receiver buffer
        call uconecho      ;read upper case console character
        cmp al,'C'
        je wboot_1         ;cancel
        cmp al,'R'
        je outer_retry     ;retry 10 more times
        cmp al,'I'
        je z_ret           ;ignore error
        or al,255          ;set code for permanent error
z_ret:  ret
```

```
wboot_1:                    ;can't make it w/ a short leap
        jmp WBOOT

;**********************************************
;*                                            *
;*  sendcom sends the address of the iopb to  *
;*    the iSBC 201                            *
;*                                            *
;**********************************************

sendcom:
        MOV CL,4
        MOV AX,DMA_SEG
        SAL AX,CL
        ADD AX,DMA_ADR
        MOV IO_ADR,AX
        MOV CL,4
        MOV AX,CS
        SAL AX,CL
        ADD AX,OFFSET CHANCMD ;ADD SEG & OFFSET FOR 201
        out ilow,al
        mov cl,8
        sar ax,cl
        out ihigh,al
        ret

;********************************************
;*                                          *
;*                Data Areas                *
;*                                          *
;********************************************
data_offset     equ offset $

        dseg
        org     data_offset     ;contiguous with code segment
IOBYTE  db      0
disk    db      0       ;disk number
chancmd db      80h     ;iopb channel word
io_com  db      0
nsec    db      1       ;number sectors to xfer
trk     db      0
sect    db      0       ;start sector
IO_ADR  DW      0000H   ;PHYS ADDR FOR SBC201 USE
dma_adr dw      0080h   ;DMA adr (default)
dma_seg dw      0       ;DMA Base Segment

HOM_COM EQU 3
RDCODE  EQU 4
```

```
ERR_CODE DB ØØH
VRCODE  EQU 6

        IF      loader_bios
;-----------------------------------------------
;|                                             |
signon  db      cr,lf,cr,lf
        db      'CP/M-86 Version 1.Ø',cr,lf,Ø
;|                                             |
;-----------------------------------------------
        ENDIF   ;loader_bios

        IF      not loader_bios
;-----------------------------------------------
;|                                             |
signon  db      cr,lf,cr,lf
        db      'System Generated Ø4/28/81'
        db      cr,lf,Ø
;|                                             |
;-----------------------------------------------
        ENDIF   ;not loader_bios

int_trp db      cr,lf
        db      'Interrupt Trap Halt'
        db      cr,lf,Ø

errtbl  dw erØ,er1,er2,er3
        dw er4,er5,er6,er7
        dw er8,er9,erA,erB
        dw erC,erD,erE,erF
        dw er1Ø,er2Ø,er4Ø,er8Ø

erØ     db  cr,lf,'Null Error ??',Ø
er1     db  cr,lf,'Deleted Record :',Ø
er2     db  cr,lf,'CRC Error :',Ø
er3     db  cr,lf,'Data Overrun-Underrun :',Ø
er4     db  cr,lf,'Seek Error :',Ø
er5     equ erØ
er6     equ erØ
er7     equ erØ
er8     db  cr,lf,'Address Error :',Ø
er9     db  cr,lf,'Write Protect :',Ø
erA     db  cr,lf,'ID CRC Error :',Ø
erB     db  cr,lf,'Write Error :',Ø
erC     db  cr,lf,'Sector Not Found :',Ø
erD     equ erØ
erE     db  cr,lf,'No Address Mark :',Ø
erF     db  cr,lf,'Data Mark Error :',Ø
er1Ø    equ er3
er2Ø    equ er9
er4Ø    equ erB
```

```
er80     db  cr,lf,'Drive Not Ready :',0

rtry_cnt db 0    ;disk error retry counter

;        System Memory Segment Table

segtable db 1    ;1 segments
         dw tpa_seg       ;1st seg starts after BIOS
         dw tpa_len       ;and extends to 28000

         include singles.lib ;read in disk definitions

loc_stk rw  32  ;local stack for initialization
stkbase equ offset $

lastoff equ offset $
tpa_seg equ (lastoff+0400h+15) / 16
tpa_len equ 0F00h - tpa_seg
         db 0     ;fill last address for GENCMD

;*********************************************
;*                                           *
;*          Dummy Data Section               *
;*                                           *
;*********************************************
         dseg     0       ;absolute low memory
         org      0       ;(interrupt vectors)
int0_offset       rw      1
int0_segment      rw      1
;        pad to system call vector
         rw       2*(bdos_int-1)

bdos_offset       rw      1
bdos_segment      rw      1
         END
```

rtry_cnt db 0   ;disk error retry counter

APPENDIX B

```
        title   'Customized Basic I/O System'

;*********************************************
;*                                           *
;* This Customized BIOS adapts CP/M-86 to    *
;* the following hardware configuration      *
;*      Processor:  iSBC 8612                *
;*      Controller:  iSBC 202                *
;*      Memory model:  8080                  *
;*                                           *
;*      Programmer:  M.B. Candalor           *
;*      Revisions :                          *
;*                                           *
;*********************************************

true                equ -1
false               equ not true
cr                  equ 0dh ;carriage return
lf                  equ 0ah ;line feed
max_retries         equ 10  ;for disk i/o, before perm error

;*********************************************
;*                                           *
;* Loader_bios is true if assembling the     *
;* LOADER BIOS, otherwise BIOS is for the    *
;* CPM.SYS file.                             *
;*                                           *
;*********************************************

LOADER_BIOS         EQU TRUE
bdos_int            equ 224 ;reserved BDOS interrupt

        IF          not loader_bios
;---------------------------------------------
;|                                           |
bios_code           equ 2500h
ccp_offset          equ 0000h
bdos_ofst           equ 0B06h ;BDOS entry point
;|                                           |
;---------------------------------------------
        ENDIF       ;not loader_bios

        IF          loader_bios
;---------------------------------------------
;|                                           |
bios_code           equ 1200h ;start of LDBIOS
ccp_offset          equ 0003h ;base of CPMLOADER
bdos_ofst           equ 0406h ;stripped BDOS entry
```

```
;|                                             |
;-----------------------------------------------
        ENDIF   ;loader_bios

csts    equ 0dah          ;I8251 status port
cdata   equ 0d8h          ;  "      data   "
;
;
;***********************************************
;*                                             *
;* INTEL iSBC 202 Disk Controller Ports        *
;*                                             *
;***********************************************


base    equ     078h
rtype   equ     base+1
rbyte   equ     base+3
reset   equ     base+7

dstat   equ     base
ilow    equ     base+1
ihigh   equ     base+2

        cseg
        org     ccpoffset
ccp:
        org     bios_code

;***********************************************
;*                                             *
;* BIOS Jump Vector for Individual Routines    *
;*                                             *
;***********************************************

 jmp INIT          ;Enter from BOOT ROM or LOADER
 jmp WBOOT         ;Arrive here from BDOS call 0
 jmp CONST         ;return console keyboard status
 jmp CONIN         ;return console keyboard char
 jmp CONOUT        ;write char to console device
 jmp LISTOUT       ;write character to list device
 jmp PUNCH         ;write character to punch device
 jmp READER        ;return char from reader device
 jmp HOME          ;move to trk 00 on cur sel drive
 jmp SELDSK        ;select disk for next rd/write
 jmp SETTRK        ;set track for next rd/write
 jmp SETSEC        ;set sector for next rd/write
 jmp SETDMA        ;set offset for user buff (DMA)
 jmp READ          ;read a 128 byte sector
 jmp WRITE         ;write a 128 byte sector
 jmp LISTST        ;return list status
```

```
 jmp SECTRAN      ;xlate logical->physical sector
 jmp SETDMAB      ;set seg base for buff (DMA)
 jmp GETSEGT      ;return offset of Mem Desc Table
 jmp GETIOBF      ;return I/O map byte (IOBYTE)
 jmp SETIOBF      ;set I/O map byte (IOBYTE)

;**********************************************
;*                                            *
;* INIT Entry Point, Differs for LDBIOS and   *
;* BIOS, according to "Loader_Bios" value     *
;*                                            *
;**********************************************

INIT:   ;print signon message and initialize hardware
        mov ax,cs           ;we entered with a JMPF so use
        mov ss,ax           ;CS: as the initial value of SS:,
        mov ds,ax           ;DS:,
        mov es,ax           ;and ES:
        ;use local stack during initialization
        mov sp,offset stkbase
        cld                 ;set forward direction

        IF      not loader_bios
;---------------------------------------------
;|                                           |
        ; This is a BIOS for the CPM.SYS file.
        ; Setup all interrupt vectors in low
        ; memory to address trap

        push ds             ;save the DS register
        mov IOBYTE,0        ;clear IOBYTE
        mov ax,0
        mov ds,ax
        mov es,ax           ;set ES and DS to zero
        ;setup interrupt 0 to address trap routine
        mov int0_offset,offset int_trap
        mov int0_segment,CS
        mov di,4
        mov si,0            ;then propagate
        mov cx,510          ;trap vector to
        rep movs ax,ax      ;all 256 interrupts
        ;BDOS offset to proper interrupt
        mov bdos_offset,bdos_ofst
        pop ds              ;restore the DS register

;       (additional CP/M-86 initialization)
;|                                           |
;---------------------------------------------
        ENDIF   ;not loader_bios

        IF      loader_bios
```

```
;-------------------------------------------
;|                                         |
        ;This is a BIOS for the LOADER
        push ds         ;save data segment
        mov ax,0
        mov ds,ax       ;point to segment zero
        ;BDOS interrupt offset
        mov bdos_offset,bdos_ofst
        mov bdos_segment,CS ;bdos interrupt segment
;       (additional LOADER initialization)
        pop ds          ;restore data segment
;|                                         |
;-------------------------------------------
        ENDIF   ;loader_bios

        mov bx,offset signon
        call pmsg       ;print signon message
        mov cl,0        ;default to dr A: on coldstart
        jmp ccp         ;jump to cold start entry of CCP

WBOOT:  jmp ccp+6       ;direct entry to CCP at command level

        IF      not loader_bios
;-------------------------------------------
;|                                         |
int_trap:
        cli             ;block interrupts
        mov ax,cs
        mov ds,ax       ;get our data segment
        mov bx,offset int_trp
        call pmsg
        hlt             ;hardstop
;|                                         |
;-------------------------------------------
        ENDIF   ;not loader_bios

;*******************************************
;*                                         *
;*   CP/M Character I/O Interface Routines *
;*                                         *
;*    console is USART (i8251A) on iSBC 8612 *
;*               at ports D8/DA            *
;*******************************************

CONST:          ;console status
        in al,csts
        and al,2
        jz const_ret
        or al,255       ;return non-zero if rda
const_ret:
        ret             ;rcvr data available
```

```
CONIN:                          ;console input
        call CONST
        jz CONIN                ;wait for RDA
        in al,cdata
        and al,7fh              ;read data & remove parity bit
        ret

CONOUT:         ;console output
        in al,csts
        and al,1                ;get console status
        jz CONOUT
        mov al,cl
        out cdata,al            ;transmitter buffer is empty
        ret                     ;then return data

LISTOUT:                        ;list device output
                                ;not implemented
        ret

LISTST:                         ;poll list status
                                ;not implemented
        ret

PUNCH:          ;write punch device
                                ;not implemented

READER:
        mov al,1ah              ;return eof for now
        ret

GETIOBF:
        MOV AL,0                ;IOBYTE NOT IMPLEMENTED
        ret

SETIOBF:
        ret                     ;iobyte not implemented

; Routine to get and echo a console character
;       and shift it to upper case

uconecho:
        call CONIN              ;get a console character
        push ax
        mov cl,al               ;save and
        call CONOUT
        pop ax                  ;echo to console
        cmp al,'a'
        jb uret                 ;less than 'a' is ok
        cmp al,'z'
        ja uret                 ;greater than 'z' is ok
```

```
        sub al,'a'-'A'  ;else shift to caps
uret:
        ret
pmsg:
        mov al,[BX]     ;get next char from message
        test al,al
        jz return       ;if zero return
        mov CL,AL
        call CONOUT     ;print it
        inc BX
        jmps pmsg       ;next character and loop

;*********************************************
;*                                           *
;*          Disk Input/Output Routines       *
;*                                           *
;*********************************************

SELDSK:         ;select disk given by register CL
ndisks  equu    2 ;number of disks (up to 16)
        mov disk,cl     ;save disk number
        mov bx,0000h    ;ready for error return
        cmp cl,ndisks   ;n beyond max disks?
        jnb return      ;return if so
        mov ch,0        ;double(n)
        mov bx,cx       ;bx = n
        mov cl,4        ;ready for *16
        shl bx,cl       ;n = n * 16
        mov cx,offset dpbase
        add bx,cx       ;dpbase + n * 16
return: ret             ;bx = .dph

HOME:   ;move selected disk to home position (Track 0)
        mov io_com,homcom
        mov trk,0
        call execute
        ret

SETTRK: ;set track address given by CL
        mov trk,CL
        ret

SETSEC: ;set sector number given by cl
        mov sect,CL
        ret

SECTRAN: ;translate sector CX using table at [DX]
        mov ch,0
        mov bx,cx
        TEST DX,00      ;IS THERE A SKEW?
        JZ NO_SKEW      ;IF NOT, RET
```

```
        add bx,dx           ;add sector to tran table address
        mov bl,[bx]         ;get logical sector
        ret
NO_SKEW:
        ADD BX,1
        RET

SETDMA: ;set DMA offset given by CX
        mov dma_adr,CX
        ret

SETDMAB: ;set DMA segment given by CX
        mov dma_seg,CX
        ret
;
GETSEGT:  ;return address of physical memory table
        mov bx,offset seg_table
        ret

;*********************************************************
;*                                                       *
;*  All disk I/O parameters are setup:                   *
;*     DISK     is disk number       (SELDSK) *
;*     TRK      is track number      (SETTRK) *
;*     SECT     is sector number     (SETSEC) *
;*      IO_ADR  IS THE PHYS ADDR FOR DMA      *
;*     DMA_ADR  is the DMA 1st offset         *
;*  READ reads the selected sector to the DMA*
;*  address, and WRITE writes the data from  *
;*  the DMA address to the selected sector   *
;*                                           *
;*********************************************************

READ:
        mov cl,4
        mov al,disk         ;combine disk selection
        sal al,cl           ;with opcode
        or al,rdcode
        mov io_com,al       ;create iopb
        jmps execute

WRITE:
        mov cl,4
        mov al,disk
        sal al,cl
        or al,wrcode        ;create iopb for write
        mov io_com,al

EXECUTE:

outer_retry:
```

```
        mov rtry_cnt,max_retries

retry:
        in al,rtype      ;clear controller
        in al,rbyte      ;
        call sendcom

idle:   in al,istat      ;wait for completion
        and al,4         ;ready
        jz idle

;       check i.o. completion ok
        in al,rtype
;       00 unlinked i/o complete        01 linked i/o comp
;       10 disk status changed          11 (not used)
;       must be a 00 in al
        test al,10b      ;ready status change?
        JNZ WREADY
        OR AL,0
        jnz werror       ;some other error, retry

;       check i/o error bits
        in al,rbyte
        rcl al,1
        mov err_code,80h
        jb wready        ;unit not ready
        rcr al,1
        mov err_code,al
        and al,0feh      ;any other errors?
        jnz werror
;
;       read or write is ok, al contains 0
        ret

wready: ;not ready, treat as an error for now
        in al,rbyte      ;clear result byte
        jmps trycount

werror: ;return hardware malfunction
trycount:
        dec rtry_cnt
        jnz retry
        mov al,err_code
        mov ah,0
        mov bx,ax        ;make error code 16 bits
        mov bx,errtbl[BX]
        call pmsg        ;print appropriate message
        in al,cdata      ;flush usart receiver buffer
        call uconecho    ;read upper case console character
        cmp al,'C'
        je wboot_1       ;cancel
```

```
        cmp al,'R'
        je outer_retry  ;retry 10 more times
        cmp al,'I'
        je z_ret        ;ignore error
        or al,255       ;set code for permanent error
z_ret:  ret


whoot_l:                ;can't make it w/ a short leap
        jmp WBOOT

;*********************************************************
;*                                                       *
;*  sendcom sends the address of the iopb to             *
;*    the iSBC 202                                       *
;*                                                       *
;*********************************************************

sendcom:
        MOV CL,4
        MOV AX,DMA_SEG
        SAL AX,CL
        ADD AX,DMA_ADR
        MOV IO_ADR,AX
        MOV CL,4
        MOV AX,CS
        SAL AX,CL
        ADD AX,OFFSET CHANCMD ;ADD SEG & OFFSET FOR 202
        out ilow,al
        mov cl,8
        sar ax,cl
        out ihigh,al
        ret

;************************************************
;*                                              *
;*               Data Areas                     *
;*                                              *
;************************************************
data_offset     equ offset $

        dseg
        org     data_offset     ;contiguous with code segment
IOBYTE  db      0
disk    db      0       ;disk number
chancmd db      80h     ;iopb channel word
io_com  db      0
nsec    db      1       ;number sectors to xfer
trk     db      0
sect    db      0       ;start sector
```

```
IO_ADR   DW       0000H   ;PHYS ADDR FOR SBC202 USE
dma_adr dw        0080h   ;DMA adr (default)
dma_seg dw        0       ;DMA Base Segment

HOM_COM EQU 3
RDCODE  ECU 4
ERR_CODE DB 00H
WRCODE  ECU 6

        IF      loader_bios
;-------------------------------------------
;|                                         |
signon  db      cr,lf,cr,lf
        db      'CP/M-86 Version 1.0',cr,lf,0
;|                                         |
;-------------------------------------------
        ENDIF   ;loader_bios

        IF      not loader_bios
;-------------------------------------------
;|                                         |
signon  db      cr,lf,cr,lf
        db      'System Generated 05/25/81'
        db      cr,lf,0
;|                                         |
;-------------------------------------------
        ENDIF   ;not loader_bios

int_trp db      cr,lf
        db      'Interrupt Trap Halt'
        db      cr,lf,0

errtbl  dw er0,er1,er2,er3
        dw er4,er5,er6,er7
        dw er8,er9,erA,erB
        dw erC,erD,erE,erF
        dw er10,er20,er40,er80

er0     db  cr,lf,'Null Error ??',0
er1     db  cr,lf,'Deleted Record :',0
er2     db  cr,lf,'CRC Error :',0
er3     db  cr,lf,'Data Overrun-Underrun :',0
er4     db  cr,lf,'Seek Error :',0
er5     equ er0
er6     equ er0
er7     equ er0
er8     db  cr,lf,'Address Error :',0
er9     db  cr,lf,'Write Protect :',0
erA     db  cr,lf,'ID CRC Error :',0
erB     db  cr,lf,'Write Error :',0
erC     db  cr,lf,'Sector Not Found :',0
```

```
erD      equ er0
erE      db   cr,lf,'No Address Mark :',0
erF      db   cr,lf,'Data Mark Error :',0
er10     equ er3
er20     equ er9
er40     equ erB
er80     db   cr,lf,'Drive Not Ready :',0

rtry_cnt db 0    ;disk error retry counter

;        System Memory Segment Table

segtable db 1    ;1 segments
         dw tpa_seg       ;1st seg starts after BIOS
         dw tpa_len       ;and extends to 08000

         INCLUDE DOUBLE.LIB       ;READ IN DISK DEFINITIONS

loc_stk rw  32  ;local stack for initialization
stkbase equ offset $

lastoff equ offset $
tpa_seg equ (lastoff+0400h+15) / 16
tpa_len equ 0F00h - tpa_seg
         db 0     ;fill last address for GENCMD

;*********************************************************
;*                                                       *
;*          Dummy Data Section                           *
;*                                                       *
;*********************************************************
         dseg     0         ;absolute low memory
         org      0         ;(interrupt vectors)
int0_offset       rw        1
int0_segment      rw        1
;        pad to system call vector
         rw       2*(bdos_int-1)

bdos_offset       rw        1
bdos_segment      rw        1
         END

rtry_cnt db 0    ;disk error retry counter
```

APPENDIX C

```
;
; ROM bootstrap for CP/M-86 on an iSBC86/12
;                    with the
;  iSBC 201 & 202 Floppy Disk Controllers
;
;
;        Copyright (C) 1980,1981
;        Digital Research, Inc.
;        Box 579, Pacific Grove
;        California, 93950
;
;*********************************************
;* This is the BOOT ROM which is resident   *
;* in the 957 monitor. To execute the boot  *
;* the monitor must be brought on-line and  *
;* then control passed by the command       *
;* "effd4:0".  First, the ROM moves         *
;* a copy of its data area to RAM at loca-  *
;* tion 00000H, then initializes the segment*
;* registers and the stack pointer.  The    *
;* various peripheral interface chips on the*
;* SBC 86/12 are initialized.  The 8251     *
;* serial interface is configured for a 9600*
;* baud asynchronous terminal, and the in-  *
;* terrupt controller is setup for inter-   *
;* rupts 10H-17H (vectors at 00040H-0005FH) *
;* and edge-triggered auto-EOI (end of in-  *
;* terrupt) mode with all interrupt levels  *
;* masked-off.  Next, the 201-202 Diskette  *
;* controller is initialized, and track 0   *
;* sector 1 is read to determine the target *
;* paragraph address for LOADER.  Finally,  *
;* the LOADER on track 0 sectors 2-26 and   *
;* track 1 sectors 1-26 is read into the    *
;* target address.  Control then transfers  *
;* to LOADER.  ROM                          *
;* 0 contains the even memory locations, and*
;* ROM 1 contains the odd addresses.  BOOT  *
;* ROM uses RAM between 00000H and 000FFH   *
;* (absolute) for a scratch area, along with*
;* the sector 1 buffer.                     *
;*********************************************
;
cr              equu    13
lf              equ     10
;
;       disk ports and commands
;
```

```
base            equ     278h
rtype           equ     base+1
rbyte           equ     base+3
reset           equ     base+7
;
istat           equ     base
ilow            equ     base+1
ihigh           equ     base+2
;
;actual console baud rate
baud_rate       equ     9600
;value for 8253 baud counter
baud            equ     768/(baud_rate/100)
;
csts            equ     0DAh    ;i8251 status port
cdata           equ     0D8h    ;  "     data port
;
tch0            equ     0D0h    ;8253 PIC channel 0 port
tch1            equ     tch0+2  ;ch 1 port
tch2            equ     tch0+4  ;ch 2 port
tcmd            equ     tch0+6  ;8253 command port
;
icp1            equ     0C0h    ;8259a port 0
icp2            equ     0C2h    ;8259a port 1
;
secsec          equ     0c8h    ;offset for track 1
;
ROMSEG          EQU     0FFD4H
;
;
;
        cseg    romseg
;
;First, move our data area into RAM at 0000:0200
;
        mov ax,cs
        mov ds,ax           ;point DS to CS for source
        mov SI,drombegin          ;start of data
        mov DI,offset ram_start ;offset of destination
        mov ax,0
        mov es,ax           ;destination segment is 0000
        mov CX,data_length        ;how much to move in bytes
            rep movs al,al            ;move out of eprom a byte
                                  ;at a time
;
        mov ax,0
        mov ds,ax           ;data segment now in RAM
        mov ss,ax
        mov sp,stack_offset       ;Initialize stack segment/
                                  ;pointer
        cld                       ;clear the direction flag
```

```
;
;
;Setup the 8259 Programmable Interrupt Controller
;
        mov al,13h
        out icp1,al     ;8259a ICW 1  8086 mode
        mov al,10h
        out icp2,al     ;8259a ICW 2  vector @ 40-5F
        mov al,1Fh
        out icp2,al     ;8259a ICW 4  auto EOI master
        mov al,0FFh
        out icp2,al     ;8259a OCW 1  mask all levels off
;
;Reset and initialize the 221/222 Diskette Interface
;
restart:        ;also come back here on fatal errors
        in  al,rtype    ;clear status type register
        in al,rbyte     ;clear status register
        out reset,al    ;reset diskette system
homer:  mov BX,offset home
        CALL execute    ;home drive 0
;
        mov bx,OFFSET sector1   ;offset for first sector DMA
        mov ax,bx       ;enter in packet
        mov bx,offset read0+5   ;"
        mov [bx],al             ;"
        inc bx
        mov [bx],ah             ;packet now complete
        mov bx,offset read0     ;packet location
        call execute            ;send packet
;
        mov es,abs              ;segment loc for LOADER
        mov ax,es               ;must translate to 16 bit abs
        mov cl,04               ;addr for diskette controller
        sal ax,cl
        mov bx,offset read1+5
        mov [bx],al             ;enter in packet
        inc bx
        mov [bx],ah
        mov bx,offset read1
        call execute            ;read track 0
;
        mov cl,04
        mov ax,es               ;compute offset for track 1
        add ax,secsec
        sal ax,cl
        mov bx,offset read2+5
        mov [bx],al
        inc bx
        mov [bx],ah
        mov bx,offset read2
```

```
        call execute            ;read track 2
;
        mov leap_segment,es
;
        ;setup far jump vector
        mov leap_offset,0
;
;
;
;
;       enter LOADER
        jmpf dword ptr leap_offset
;
pmsg:
        mov cl,[BX]
        test cl,cl
        jz return
        call conout
        inc BX
        jmp pmsg
;
conout:
        in al,csts
        test al,1
        jz conout
        mov al,cl
        out cdata,al
        ret
;
conin:
        in al,csts
        test al,2
        jz conin
        in al,cdata
        and al,7Fh
        ret
;
;
;
execute:
retry:                          ;ret  if drive not ready
        in al,rtype             ;clear controller
        in al,rbyte             ;   "        "
        call sendcom
idle:   in al,istat             ;system status
        and al,4
        jz idle         ;system awaiting interupt
        in al,rtype             ;check drive status
        test al,2
        jz intcmp
        JMP RETRY               ;I/O NOT COMPLETE, TRY AGAIN
```

```
intcmp: in al,rbyte                 ;io is complete get status
        rcl al,1
        jae iocmp
        RCR AL,1
        jmp fatal
iocmp:  rcr al,1                    ;restore
        and al,0fen                 ;any errors ?
        jz return
        JMP RETRY
;
;
;
fatal:                              ; fatal error
        mov cl,0
FTEST:  RCR AL,1
        inc cl
        TEST AL,01
        jz ftest
        mov al,cl
        mov ah,0
        ADD AX,AX
        mov bx,ax                   ;make 16 bits
        mov bx,errtbl[BX]
;       print appropriate error message
        call pmsg
        call conin                  ;wait for key strike
        pop ax                      ;discard unused item
        jmp restart                 ;then start all over
;
return:
        RET                         ;return from EXECUTE
;
;
;
;
sendcom:        ;routine to send a command string to 201/201
        mov ax,bx
        out ilow,al
        mov cl,08
        sar ax,cl
        out ihigh,al                ;packet addr
        ret
;
;
;
;
;       Image of data to be moved to RAM
;
drombegin equ offset $
;
;
```

```
creadstring     db      80h     ;parameter block load
                db      4h      ;read function code for drive
                db      1       ;# sectors to read
                db      0       ;track #
                db      1       ;start with sector 1
                db      0       ;will contain lower byte addr
                db      0       ;"       "      upper
;
creadtrk0       db      80h
                db      4h      ;read multiple
                db      25      ;# sectors to read
                db      0       ;track 0
                db      2       ;start with 2
                db      0       ;addr for track 0 goes here
                db      0       ;"
;
creadtrk1       db      80h
                db      4h
                db      26      ;sectors
                db      1       ;track #
                db      1       ;start with sector 1
                db      0       ;addr lsb
                db      0       ;addr msb
;
chome0          db      80h
                DB      03H
                db      0
                db      0
                db      0
                db      0
                db      0
;
cerrtbl dw      offset er0
        dw      offset er1
        dw      offset er2
        dw      offset er3
        dw      offset er4
        dw      offset er5
        dw      offset er6
        dw      offset er7
;
Cer0    db      cr,lf,'Null Error ??',0
Cer1    db      cr,lf,'CRC Error',0
Cer2    db      cr,lf,'Seek Error',0
Cer3    db      cr,lf,'Address Error',0
Cer4    db      cr,lf,'Data Overrun-Underrun',0
Cer5    db      cr,lf,'Write Protect',0
Cer6    db      cr,lf,'Write Error',0
Cer7    db      cr,lf,'Drive Not Ready',0
;
dromend equ offset $
```

```
;
data_length     equ dromend-drombegin
;
;       reserve space in RAM for data area
;       (no hex records generated here)
;
        dseg    0
        org     0200h
;
ram_start       equ     $
read0           rb      7       ;read track 0 sector 1
read1           rb      7       ;read T0 S2-26
read2           rb      7       ;read T1 S1-26
home            rb      7       ;home drive 0
errtbl          rw      8
er0             rb      length cer0     ;16
er1             rb      length cer1
er2             rb      length cer2
er3             rb      length cer3
er4             rb      length cer4     ;14
er5             rb      length cer5     ;11
er6             rb      length cer6     ;15
er7             rb      length cer7     ;17
;
leap_offset     rw      1
leap_segment    rw      1
;
;
                rw      32      ;local stack
stack_offset    equ     offset $;stack from here down
;
;               T0 S1 read in here
sector1         equ offset $
;
Ty              rb      1
Len             rw      1
Abs             rw      1       ;ABS is all we care about
Min             rw      1
Max             rw      1
                end
```

APPENDIX D

```
        title   '8086 Disk I/O Drivers'

;*****************************************************
;*                                                   *
;* Basic Input/Output System (BIOS) for              *
;* CP/M-86 Configured for iSBC 86/12 with            *
;* the iSBC 204 Floppy Disk Controller               *
;*                                                   *
;* (Note:  this file contains both embedded          *
;* tabs and blanks to minimize the list file         *
;* width for printing purposes.  You may wish*
;* to expand the blanks before performing            *
;* major editing.)                                   *
;*****************************************************

;       Copyright (C) 1980,1981
;       Digital Research, Inc.
;       Box 579, Pacific Grove
;       California, 93950
;
;       (Permission is hereby granted to use
;       or abstract the following program in
;       the implementation of CP/M, MP/M or
;       CP/NET for the 8086 or 8088 Micro-
;       processor)

true            equ -1
false           equ not true

;*****************************************************
;*                                                   *
;* Loader_bios is true if assembling the             *
;* LOADER BIOS, otherwise BIOS is for the            *
;* CPM.SYS file.  Blc_list is true if we             *
;* have a serial printer attached to BLC8538 *
;* Bdos_int is interrupt used for earlier            *
;* versions.                                         *
;*                                                   *
;*****************************************************

loader_bios     equ false
blc_list        equ true
bdos_int        equ 224 ;reserved BDOS Interrupt

        IF      not loader_bios
;---------------------------------------------
;|                                           |
```

```
bios_code           equ 2500h
ccp_offset          equ 0000h
bdos_ofst           equ 0B06h ;BDOS entry point
;|                                        |
;-----------------------------------------
        ENDIF   ;not loader_bios

        IF      loader_bios
;-----------------------------------------
;|                                        |
bios_code           equ 1200h ;start of LDBIOS
ccp_offset          equ 0003h ;base of CPMLOADER
bdos_ofst           equ 0406h ;stripped BDOS entry
;|                                        |
;-----------------------------------------
        ENDIF   ;loader_bios

csts                equ 0DAh   ;i8251 status port
cdata               equ 0D8h   ;   "   data port

        IF      blc_list
;-----------------------------------------
;|                                        |
lsts                equ 41h ;2651 No. 0 on BLC8538 status port
ldata               equ 40h ;  "   "  "  "   "     data port
blc_reset           equ 60h ;reset selected USARTS on BLC8538
;|                                        |
;-----------------------------------------
        ENDIF   ;blc_list

;*****************************************
;*                                       *
;*    Intel iSBC 204 Disk Controller Ports   *
;*                                       *
;*****************************************

base204             equ 0a0h            ;SBC204 assigned address

fdc_com             equ base204+0       ;8271 FDC out command
fdc_stat            equ base204+0       ;8271 in status
fdc_parm            equ base204+1       ;8271 out parameter
fdc_rslt            equ base204+1       ;8271 in result
fdc_rst             equ base204+2       ;8271 out reset
dmac_adr            equ base204+4       ;8257 DMA base address out
dmac_cont           equ base204+5       ;8257 out control
dmac_scan           equ base204+6       ;8257 out scan control
dmac_sadr           equ base204+7       ;8257 out scan address
dmac_mode           equ base204+8       ;8257 out mode
dmac_stat           equ base204+8       ;8257 in status
fdc_sel             equ base204+9       ;FDC select port (not used)
fdc_segment         equ base204+10      ;segment address register
```

```
reset_204        equ base204+15  ;reset entire interface

max_retries      equ 10          ;max retries on disk i/o
                                 ;before perm error
cr               equ 0dh         ;carriage return
lf               equ 0ah         ;line feed

         cseg
         org     ccpoffset
ccp:
         org     bios_code

;*********************************************
;*                                           *
;* BIOS Jump Vector for Individual Routines  *
;*                                           *
;*********************************************

 jmp INIT         ;Enter from BOOT ROM or LOADER
 jmp WBOOT        ;Arrive here from BDOS call 0
 jmp CONST        ;return console keyboard status
 jmp CONIN        ;return console keyboard char
 jmp CONOUT       ;write char to console device
 jmp LISTOUT      ;write character to list device
 jmp PUNCH        ;write character to punch device
 jmp READER       ;return char from reader device
 jmp HOME         ;move to trk 00 on cur sel drive
 jmp SELDSK       ;select disk for next rd/write
 jmp SETTRK       ;set track for next rd/write
 jmp SETSEC       ;set sector for next rd/write
 jmp SETDMA       ;set offset for user buff (DMA)
 jmp READ         ;read a 128 byte sector
 jmp WRITE        ;write a 128 byte sector
 jmp LISTST       ;return list status
 jmp SECTRAN      ;xlate logical->physical sector
 jmp SETDMAB      ;set seg base for buff (DMA)
 jmp GETSEGT      ;return offset of Mem Desc Table
 jmp GETIOBF      ;return I/O map byte (IOBYTE)
 jmp SETIOBF      ;set I/O map byte (IOBYTE)

;*********************************************
;*                                           *
;* INIT Entry Point, Differs for LDBIOS and  *
;* BIOS, according to "Loader_Bios" value    *
;*                                           *
;*********************************************

INIT:    ;print signon message and initialize hardware
         mov ax,cs        ;we entered with a JMPF so use
         mov ss,ax        ; CS: as the initial value of SS:,
         mov ds,ax        ;        DS:,
```

```
        mov es,ax           ;        and ES:
        ;use local stack during initialization
        mov sp,offset stkbase
        cld                 ;set forward direction

        IF      not loader_bios
;-----------------------------------------------
;|                                             |
        ; This is a BIOS for the CPM.SYS file.
        ; Setup all interrupt vectors in low
        ; memory to address trap

        push ds             ;save the DS register
        mov ax,0
        mov ds,ax
        mov es,ax           ;set ES and DS to zero
        ;setup interrupt 0 to address trap routine
        mov int0_offset,offset int_trap
        mov int0_segment,CS
        mov di,4
        mov si,0            ;then propagate
        mov cx,510          ;trap vector to
        rep movs ax,ax      ;all 256 interrupts
        ;BDOS offset to proper interrupt
        mov bdos_offset,bdos_ofst
        pop ds              ;restore the DS register

;***********************************************
;*                                             *
;* National "BLC 8538" Channel 0 for a serial*
;* 9600 baud printer - this board uses 8 Sig-*
;* netics 2651 Usarts which have on-chip baud*
;* rate generators.                            *
;*                                             *
;***********************************************

        mov al,0FFh
        out blc_reset,al ;reset all usarts on 8538
        mov al,4Eh
        out ldata+2,al  ;set usart 0 in async 8 bit mode
        mov al,3Eh
        out ldata+2,al  ;set usart 0 to 9600 baud
        mov al,37h
        out ldata+3,al  ;enable Tx/Rx, and set up RTS,DTR
;|                                             |
;-----------------------------------------------
        ENDIF   ;not loader_bios

        IF      loader_bios
;-----------------------------------------------
;|                                             |
```

```
        ;This is a BIOS for the LOADER
        push ds             ;save data segment
        mov ax,0
        mov ds,ax           ;point to segment zero
        ;BDOS interrupt offset
        mov bdos_offset,bdos_ofst
        mov bdos_segment,CS ;bdos interrupt segment
        pop ds              ;restore data segment
;|                                                  |
;---------------------------------------------------
        ENDIF   ;loader_bios

        mov bx,offset signon
        call pmsg           ;print signon message
        mov cl,0            ;default to dr A: on coldstart
        jmp ccp             ;jump to cold start entry of CCP

WBOOT:  jmp ccp+6           ;direct entry to CCP at command level

        IF      not loader_bios
;---------------------------------------------------
;|                                                  |
int_trap:
        cli                 ;block interrupts
        mov ax,cs
        mov ds,ax           ;get our data segment
        mov bx,offset int_trp
        call pmsg
        hlt                 ;hardstop
;|                                                  |
;---------------------------------------------------
        ENDIF   ;not loader_bios

;*************************************************
;*                                               *
;*   CP/M Character I/O Interface Routines       *
;*   Console is Usart (i8251a) on iSBC 86/12     *
;*   at ports D8/DA                              *
;*                                               *
;*************************************************

CONST:          ;console status
        in al,csts
        and al,2
        jz const_ret
        or al,255           ;return non-zero if RDA
const_ret:
        ret                 ;Receiver Data Available

CONIN:                      ;console input
        call const
```

```
        jz CONIN            ;wait for RDA
        in al,cdata
        and al,7fh          ;read data and remove parity bit
        ret

CONOUT:         ;console output
        in al,csts
        and al,1            ;get console status
        jz CONOUT           ;wait for TBE
        mov al,cl
        out cdata,al        ;Transmitter Buffer Empty
        ret                 ;then return data

LISTOUT:                    ;list device output

        IF      blc_list
;-------------------------------------------
;|                                         |
        call LISTST
        jz LISTOUT          ;wait for printer not busy
        mov al,cl
        out ldata,al        ;send char to TI 810
;|                                         |
;-------------------------------------------
        ENDIF   ;blc_list

        ret

LISTST:                     ;poll list status

        IF      blc_list
;-------------------------------------------
;|                                         |
        in al,lsts
        and al,81h          ;look at both TxRDY and DTR
        cmp al,81h
        jnz zero_ret        ;either false, printer is busy
        or al,255           ;both true, LPT is ready
;|                                         |
;-------------------------------------------
        ENDIF   ;blc_list

        ret

PUNCH:  ;not implemented in this configuration
READER:
        mov al,1ah
        ret                 ;return EOF for now

GETIOBF:
        mov al,0            ;TTY: for consistency
```

```
        ret                 ;IOBYTE not implemented

SETIOBF:
        ret                 ;iobyte not implemented

zero_ret:
        and al,0
ret                         ;return zero in AL and flags

; Routine to get and echo a console character
;       and shift it to upper case

uconecho:
        call CONIN          ;get a console character
        push ax
        mov cl,al           ;save and
        call CONOUT
        pop ax              ;echo to console
        cmp al,'a'
        jb uret             ;less than 'a' is ok
        cmp al,'z'
        ja uret             ;greater than 'z' is ok
        sub al,'a'-'A'      ;else shift to caps
uret:
        ret

;       utility subroutine to print messages

pmsg:
        mov al,[BX]         ;get next char from message
        test al,al
        jz return           ;if zero return
        mov CL,AL
        call CONOUT         ;print it
        inc BX
        jmps pmsg           ;next character and loop

;*********************************************
;*                                           *
;*        Disk Input/Output Routines         *
;*                                           *
;*********************************************

SELDSK:         ;select disk given by register CL
        mov bx,0000h
        cmp cl,2            ;this BIOS only supports 2 disks
        jnb return          ;return w/ 0000 in BX if bad drive
        mov al, 80h
        cmp cl,0
        jne sel1            ;drive 1 if not zero
        mov al, 40h         ;else drive is 0
```

```
sel1:   mov sel_mask,al ;save drive select mask
                        ;now, we need disk parameter address
        mov ch,0
        mov bx,cx       ;BX = word(CL)
        mov cl,4
        shl bx,cl       ;multiply drive code * 16
        ;create offset from Disk Parameter Base
        add bx,offset dp_base
return:
        ret

HOME:   ;move selected disk to home position (Track 0)
        mov trk,0       ;set disk i/o to track zero
        mov bx,offset hom_com
        call execute
        jz return       ;home drive and return if OK
        mov bx,offset bad_hom  ;else print
        call pmsg       ;"Home Error"
        jmps home       ;and retry

SETTRK: ;set track address given by CX
        mov trk,cl      ;we only use 8 bits of track address
        ret

SETSEC: ;set sector number given by cx
        mov sect,cl     ;we only use 8 bits of sector address
        ret

SECTRAN: ;translate sector CX using table at [DX]
        mov bx,cx
        add bx,dx       ;add sector to tran table address
        mov bl,[bx]     ;get logical sector
        ret

SETDMA: ;set DMA offset given by CX
        mov dma_adr,CX
        ret

SETDMAB: ;set DMA segment given by CX
        mov dma_seg,CX
        ret
;
GETSEGT:  ;return address of physical memory table
        mov bx,offset seg_table
        ret

;*****************************************
;*                                       *
;*  All disk I/O parameters are setup:  the  *
;*  Read and Write entry points transfer one *
;*  sector of 128 bytes to/from the current  *
```

```
;*  DMA address using the current disk drive *
;*                                           *
;*********************************************

READ:
        mov al,12h       ;basic read sector command
        jmps r_w_common

WRITE:
        mov al,0ah       ;basic write sector command

r_w_common:
        mov bx,offset io_com ;point to command string
        mov byte ptr 1[BX],al ;put command into string
;       fall into execute and return

execute:  ;execute command string.
        ;[BX] points to length,
        ;       followed by Command byte,
        ;       followed by length-1 parameter bytes

        mov last_com,BX ;save command address for retries
outer_retry:
        ;allow some retrying
        mov rtry_cnt,max_retries
retry:
        mov BX,last_com
        call send_com   ;transmit command to i8271
;       check status poll

        mov BX,last_com
        mov al,1[bx]     ;get command op code
        mov cx,0800h     ;mask if it will be "int req"
        cmp al,2ch
        jb exec_poll     ;ok if it is an interrupt type
        mov cx,8080h     ;else we use "not command busy"
        and al,0fh
        cmp al,0ch       ;unless there isn't
        mov al,0
        ja exec_exit     ;       any result
                         ;poll for bits in CH,
exec_poll:               ;  toggled with bits in CL

        in al,fdc_stat   ;read status
        and al,ch
        xor al,cl        ;  isolate what we want to poll
        jz exec_poll     ;and loop until it is done

                         ;Operation complete,
        in al,fdc_rslt   ; see if result code indicates error
        and al,1eh
```

```
        jz exec_exit    ;no error, then exit
                        ;some type of error occurred . . .
        cmp al,10h
        je dr_nrdy      ;was it a not ready drive ?
                        ;no,
dr_rdy: ; then we just retry read or write
        dec rtry_cnt
        jnz retry       ;  up to 10 times

;       retries do not recover from the
;       hard error

        mov ah,0
        mov bx,ax       ;make error code 16 bits
        mov bx,errtbl[BX]
        call pmsg       ;print appropriate message
        in al,cdata     ;flush usart receiver buffer
        call uconecho   ;read upper case console character
        cmp al,'C'
        je wboot_1      ;cancel
        cmp al,'R'
        je outer_retry  ;retry 10 more times
        cmp al,'I'
        je z_ret        ;ignore error
        or al,255       ;set code for permanent error
exec_exit:
        ret

dr_nrdy:        ;here to wait for drive ready
        call test_ready
        jnz retry       ;if it's ready now we are done
        call test_ready
        jnz retry       ;if not ready twice in row,
        mov bx,offset nrdymsg
        call pmsg ;"Drive Not Ready"
nrdy01:
        call test_ready
        jz nrdy01       ;now loop until drive ready
        jmps retry      ;then go retry without decrement
zret:
        and al,0
        ret             ;return with no error code

wboot_1:                ;can't make it w/ a short leap
        jmp WBOOT

;*********************************************
;*                                           *
;*  The 18271 requires a read status command *
;*  to reset a drive-not-ready after the     *
;*  drive becomes ready                      *
```

```
;*                                                  *
;****************************************************

test_ready:
        mov dh, 40h     ;proper mask if dr 1
        test sel_mask,80h
        jnz nrdy2
        mov dh, 04h     ;mask for dr 0 status bit
nrdy2:
        mov bx,offset rds_com
        call send_com
dr_poll:
        in al,fdc_stat  ;get status word
        test al,80h
        jnz dr_poll     ;wait for not command busy
        in al,fdc_rslt  ;get "special result"
        test al,dh      ;look at bit for this drive
        ret             ;return status of ready

;****************************************************
;*                                                  *
;*  Send_com sends a command and parameters         *
;*  to the 18271:  BX addresses parameters.         *
;*  The DMA controller is also initialized          *
;*  if this is a read or write                      *
;*                                                  *
;****************************************************

send_com:
        in al,fdc_stat
        test al,80h     ;insure command not busy
        jnz send_com    ;loop until ready

        ;see if we have to initialize for a DMA operation

        mov al,1[bx]    ;get command byte
        cmp al,12h
        jne write_maybe ;if not a read it could be write
        mov cl,40h
        jmps init_dma   ;is a read command, go set DMA
write_maybe:
        cmp al,0ah
        jne dma_exit    ;leave DMA alone if not read or write
        mov cl,80h      ;we have write, not read
init_dma:
;we have a read or write operation, setup DMA controller
;       (CL contains proper direction bit)
        mov al,04h
        out dmac_mode,al    ;enable dmac
        mov al,00
        out dmac_cont,al    ;send first byte to control port
```

```
        mov al,cl
        out dmac_cont,al   ;load direction register
        mov ax,dma_adr
        out dmac_adr,al    ;send low byte of DMA
        mov al,ah
        out dmac_adr,al    ;send high byte
        mov ax,dma_seg
        out fdc_segment,al ;send low byte of segment address
        mov al,ah
        out fdc_segment,al ;then high segment address
dma_exit:
        mov cl,[BX]        ;get count
        inc BX
        mov al,[BX]        ;get command
        or al,sel_mask     ;merge command and drive code
        out fdc_com,al     ;send command byte
parm_loop:
        dec cl
        jz exec_exit       ;no (more) parameters, return
        inc BX             ;point to (next) parameter
parm_poll:
        in al,fdc_stat
        test al,20h        ;test "parameter register full" bit
        jnz parm_poll      ;idle until parm reg not full
        mov al,[BX]
        out fdc_parm,al    ;send next parameter
        jmps parm_loop     ;go see if there are more parameters

;*********************************************
;*                                           *
;*              Data Areas                   *
;*                                           *
;*********************************************
data_offset     equ offset $

        dseg
        org     data_offset     ;contiguous with code segment

        IF      loader_bios
;---------------------------------------------
;|                                           |
signon  db      cr,lf,cr,lf
        db      'CP/M-86 Version 2.2',cr,lf,0
;|                                           |
;---------------------------------------------
        ENDIF   ;loader_bios

        IF      not loader_bios
;---------------------------------------------
;|                                           |
signon  db      cr,lf,cr,lf
```

```
        db      '  System Generated  - 11 Jan 81',cr,lf,0
;|                                          |
;-------------------------------------------
        ENDIF   ;not loader_bios

bad_hom db      cr,lf,'Home Error',cr,lf,0
int_trp db      cr,lf,'Interrupt Trap Halt',cr,lf,0

errtbl  dw er0,er1,er2,er3
        dw er4,er5,er6,er7
        dw er8,er9,erA,erB
        dw erC,erD,erE,erF

er0     db  cr,lf,'Null Error ??',0
er1     equ er0
er2     equ er0
er3     equ er0
er4     db  cr,lf,'Clock Error :',0
er5     db  cr,lf,'Late DMA :',0
er6     db  cr,lf,'ID CRC Error :',0
er7     db  cr,lf,'Data CRC Error :',0
er8     db  cr,lf,'Drive Not Ready :',0
er9     db  cr,lf,'Write Protect :',0
erA     db  cr,lf,'Trk 00 Not Found :',0
erB     db  cr,lf,'Write Fault :',0
erC     db  cr,lf,'Sector Not Found :',0
erD     equ er0
erE     equ er0
erF     equ er0
nrdymsg equ er8

rtry_cnt db 0   ;disk error retry counter
last_com dw 0   ;address of last command string
dma_adr  dw 0   ;dma offset stored here
dma_seg  dw 0   ;dma segment stored here
sel_mask db 40h ;select mask, 40h or 80h

;       Various command strings for i8271

io_com  db 3    ;length
rd_wr   db 0    ;read/write function code
trk     db 0    ;track #
sect    db 0    ;sector #

hom_com db 2,29h,0      ;home drive command
rds_com db 1,2ch        ;read status command

;       System Memory Segment Table

segtable db 2   ;2 segments
        dw tpa_seg      ;1st seg starts after BIOS
```

```
        dw tpa_len          ;and extends to Ø8ØØØ
        dw 2ØØØh            ;second is 2ØØØØ -
        dw 2ØØØh            ;3FFFF (128K)

        include singles.lib ;read in disk definitions

loc_stk rw  32  ;local stack for initialization
stkbase equ offset $

lastoff equ offset $
tpa_seg equ (lastoff+Ø4ØØh+15) / 16
tpa_len equ Ø8ØØh - tpa_seg
        db Ø    ;fill last address for GENCMD

;*********************************************
;*                                           *
;*          Dummy Data Section               *
;*                                           *
;*********************************************
        dseg    Ø       ;absolute low memory
        org     Ø       ;(interrupt vectors)
intØ_offset     rw      1
intØ_segment    rw      1
;       pad to system call vector
        rw      2*(bdos_int-1)

bdos_offset     rw      1
bdos_segment    rw      1
        END
```

LIST OF REFERENCES

1. Kildall, G., "CP/M: A Family of 8- and 16-Bit Operating Systems," Byte, v. 6, p. 216-232, June 1981.

2. Digital Research, ASM-86 THE CP/M-86 ASSEMBLER USER'S GUIDE, 1981.

3. Digital Research, DDT-86 Dynamic Debugging Tool for the 8086 USER'S GUIDE, 1981.

4. Digital Research, AN INTRODUCTION TO CP/M FEATURES AND FACILITIES, 1981.

5. Digital Research, CP/M 2.2 USER'S GUIDE, 1979.

6. Digital Research, CP/M-86 SYSTEM REFERENCE GUIDE, 1981.

7. Intel Corporation, MDS-DOS HARDWARE REFERENCE MANUAL, 1976.

8. Intel Corporation, INTELLEC DOUBLE DENSITY DISKETTE OPERATING SYSTEM REFERENCE MANUAL, 1977.

9. EX-CELL-O Corporation, REMEX TECHNICAL MANUAL MULTIBUS INTERFACE ASSY 814415-001, 1980.

10. EX-CELL-O Corporation, REMEX TECHNICAL MANUAL DISKETTE DRIVE MODELS: RFD 4000-A, RFD 4001-A, RFD 2000-A, RFD 2001-A, 1979.

11. EX-CELL-O Corporation, REMEX TECHNICAL MANUAL DATA WAREHOUSE MODELS RDW 3100, RDW 3200, 1979.

AD-A104 079   NAVAL POSTGRADUATE SCHOOL MONTEREY CA   F/G 9/2
ALTERATION OF THE CP/M-86 OPERATING SYSTEM.(U)
JUN 81 M B CANDALOR

UNCLASSIFIED   NL

2 of 2
AD
A104079

END
DATE
FILMED
10-81
DTIC

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	2
4.	Associate Professor Uno R. Kodres, Code 52Kr Department of Computer Science Naval Postgraduate School Monterey, California 93940	3
5.	LCDR Robert Stilwell, SC, USN, Code 52Sb Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
6.	Professor Rudy Pannolzer, Code 62Pz Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
7.	Associate Professor Mitchell L. Cotton, Code 62Cc Department of Electrical Engineering Naval postgraduate School Monterey, California 93940	1
8.	Daniel Green, Code N-202 Naval Weapons Center Dahlgren, Virginia 22449	1
9.	PMS 400 Naval Sea Systems Command Washington, D.C. 20362 Attn: CDR Ruff	1
10.	Commander, Amphibious Squadron Five FPO San Francisco, Ca. 96601 Attn: LCDR Michael B. Candalor, USN	2